

---

# **python-bitshares Documentation**

***Release 0.1***

**Fabian Schuh**

**Jun 22, 2023**



---

## Contents

---

<b>1</b>	<b>About this Library</b>	<b>3</b>
<b>2</b>	<b>General</b>	<b>5</b>
<b>3</b>	<b>Quickstart</b>	<b>15</b>
<b>4</b>	<b>Developers and Community</b>	<b>17</b>
<b>5</b>	<b>Packages</b>	<b>19</b>
<b>6</b>	<b>Glossary</b>	<b>227</b>
<b>7</b>	<b>Indices and tables</b>	<b>229</b>
	<b>Python Module Index</b>	<b>231</b>
	<b>Index</b>	<b>233</b>



BitShares is a **blockchain-based autonomous company** (i.e. a DAC) that offers decentralized exchanging as well as sophisticated financial instruments as *products*.

It is based on *Graphene* (tm), a blockchain technology stack (i.e. software) that allows for fast transactions and ascalable blockchain solution. In case of BitShares, it comes with decentralized trading of assets as well as customized on-chain smart contracts.



# CHAPTER 1

---

## About this Library

---

The purpose of *pybitshares* is to simplify development of products and services that use the BitShares blockchain. It comes with

- it's own (bip32-encrypted) wallet
- RPC interface for the Blockchain backend
- JSON-based blockchain objects (accounts, blocks, prices, markets, etc)
- a simple to use yet powerful API
- transaction construction and signing
- push notification API
- *and more*





## 2.1 Installation

### 2.1.1 Installation

Install with *pip*:

```
$ sudo apt-get install libffi-dev libssl-dev python3-dev  
$ pip3 install bitshares
```

Manual installation:

```
$ git clone https://github.com/xeroc/python-bitshares/  
$ cd python-bitshares  
$ python3 setup.py install --user
```

### 2.1.2 Upgrade

```
$ pip3 install bitshares --user --upgrade
```

## 2.2 Quickstart

## 2.3 Tutorials

### 2.3.1 Bundle Many Operations

With BitShares, you can bundle multiple operations into a single transactions. This can be used to do a multi-send (one sender, multiple receivers), but it also allows to use any other kind of operation. The advantage here is that the

user can be sure that the operations are executed in the same order as they are added to the transaction.

```
from pprint import pprint
from bitshares import BitShares

testnet = BitShares(
    "wss://node.testnet.bitshares.eu",
    nobroadcast=True,
    bundle=True,
)

testnet.wallet.unlock("supersecret")

testnet.transfer("init0", 1, "TEST", account="xeroc")
testnet.transfer("init1", 1, "TEST", account="xeroc")
testnet.transfer("init2", 1, "TEST", account="xeroc")
testnet.transfer("init3", 1, "TEST", account="xeroc")

pprint(testnet.broadcast())
```

## 2.3.2 Proposing a Transaction

In BitShares, you can propose a transactions to any account. This is used to facilitate on-chain multisig transactions. With python-bitshares, you can do this simply by using the `proposer` attribute:

```
from pprint import pprint
from bitshares import BitShares

testnet = BitShares(
    "wss://node.testnet.bitshares.eu",
    proposer="xeroc"
)

testnet.wallet.unlock("supersecret")
pprint(testnet.transfer("init0", 1, "TEST", account="xeroc"))
```

## 2.3.3 Simple Sell Script

```
from bitshares import BitShares
from bitshares.market import Market
from bitshares.price import Price
from bitshares.amount import Amount

#
# Instantiate BitShares (pick network via API node)
#
bitshares = BitShares(
    "wss://node.testnet.bitshares.eu",
    nobroadcast=True    # <--- set this to False when you want to fire!
)

#
# Unlock the Wallet
#
bitshares.wallet.unlock("<supersecret>")
```

(continues on next page)

(continued from previous page)

```

#
# This defines the market we are looking at.
# The first asset in the first argument is the *quote*
# Sell and buy calls always refer to the *quote*
#
market = Market(
    "GOLD:USD",
    bitshares_instance=bitshares
)

#
# Sell an asset for a price with amount (quote)
#
print(market.sell(
    Price(100.0, "USD/GOLD"),
    Amount("0.01 GOLD")
))

```

### 2.3.4 Sell at a timely rate

```

import threading
from bitshares import BitShares
from bitshares.market import Market
from bitshares.price import Price
from bitshares.amount import Amount

def sell():
    """ Sell an asset for a price with amount (quote)
    """
    print(market.sell(
        Price(100.0, "USD/GOLD"),
        Amount("0.01 GOLD")
    ))

    threading.Timer(60, sell).start()

if __name__ == "__main__":
    #
    # Instanciate BitShares (pick network via API node)
    #
    bitshares = BitShares(
        "wss://node.testnet.bitshares.eu",
        nobroadcast=True # <--- set this to False when you want to fire!
    )

    #
    # Unlock the Wallet
    #
    bitshares.wallet.unlock("<supersecret>")

    #
    # This defines the market we are looking at.

```

(continues on next page)

(continued from previous page)

```
# The first asset in the first argument is the *quote*
# Sell and buy calls always refer to the *quote*
#
market = Market(
    "GOLD:USD",
    bitshares_instance=bitshares
)

sell()
```

## 2.4 Configuration

The pybitshares library comes with its own local configuration database that stores information like

- API node URL
- default account name
- the encrypted master password

and potentially more, **persistently**.

You can access those variables like a regular dictionary by using

```
from bitshares import BitShares
bitshares = BitShares()
print(bitshares.config.items())
```

Keys can be added and changed like they are for regular dictionaries.

```
bitshares.config["my-new-variable"] = "important-content"
print(bitshares.config["my-new-variable"])
```

## 2.5 FAQ

### 2.5.1 How to get order info on filled order

On CEX exchanges full order info usually available for canceled / filled orders. On BitShares such info is not available, because such info is stored in memory of bitshares-core node, and keeping non-actual orders info would take astonishing amounts of RAM.

Thus, such order info could be obtained in two ways:

- By querying account history from the node:

```
from bitshares.account import Account

a = Account('dexbot')
ops = a.history(only_ops=['fill_order'], limit=1)
for op in ops:
    print(op)
```

Note: this way has limitation: public nodes doesn't store full account history, only limited number of entries

- By querying [elasticsearch plugin](#). In short, elasticsearch plugin export account history data into elasticsearch instance, from which it's can be obtained directly or via elasticsearch wrapper. See <https://eswrapper.bitshares.eu/apidocs/> to get info on how to query the wrapper. A real-world example of elasticsearch wrapper usage for obtaining filled orders history is [bitshares-tradehistory-analyzer](#)

## 2.5.2 How to detect partially filled order

An Order have the following fields:

- `order['base']['amount']`: stores initial amount to sell
- `order['for_sale']['amount']`: stores remaining amount to sell

So, if your order initially sells 100 BTS, and 50 BTS was sold, the `order['for_sale']['amount']` will contain remaining 50 BTS.

## 2.6 Asyncio support

The library has full support of asyncio, though you need to be aware it has some limitations.

### 2.6.1 Example

A very basic example:

```
import asyncio

from bitshares.aio import BitShares
from bitshares.aio.instance import set_shared_bitshares_instance

async def info(loop, bitshares):
    await bitshares.connect()
    set_shared_bitshares_instance(bitshares)
    print(await bitshares.info())

def main():
    loop = asyncio.get_event_loop()
    bitshares = BitShares(loop=loop)
    loop.run_until_complete(info(loop, bitshares))

if __name__ == '__main__':
    main()
```

### 2.6.2 Instantiation of BitShares

To be able to perform calls, you need to explicitly run `await BitShares.connect()`. That is, creating of instance object and actual network connection are separate operations in async version:

```
bitshares = BitShares(loop=loop)
await bitshares.connect()
```

### 2.6.3 Limitations

- Most of the classes requires async init because during instantiation some API calls has to be performed:

```
await Amount('10 FOO')
```

- Several math operations are not available for `bitshares.aio.Amount`, `bitshares.aio.Price` objects. This includes multiplication, division etc. This limitation is due to inability to define python magic methods (`__mul__`, `__div__`, etc) as async coroutines
- Most of properties are awaitables too:

```
asset = await Asset('CNY')
await asset.max_market_fee
```

### 2.6.4 Subscriptions

In asyncio version subscription notifications are not handled in callback-based manner. Instead, they are available in *self.notifications* queue which is `asyncio.Queue`. You can use a single bitshares instance both for setting subscriptions and performing other API calls.

Here is the example of how to subscribe and handle events:

```
market = await Market("TEST/USD")
await bitshares.subscribe_to_market(market, event_id=4)

while True:
    event = await bitshares.notifications.get()
    print(event)
```

### 2.6.5 Debugging

To enable debugging on RPC level, you can raise loglevel on following loggers (don't forget to set formatter as well):

```
log = logging.getLogger("websockets")
log.setLevel(logging.DEBUG)

log = logging.getLogger("grapheneapi")
log.setLevel(logging.DEBUG)
```

### 2.6.6 Tests

Asyncio version has a dedicated testsuite which uses real API integration tests which are performed against local bitshares-core testnet. Bitshares node is spawned automatically inside docker container. You don't need to setup anything.

Before running tests you need to install dependencies via *pip install -r requirements-test.txt*

Run tests via *pytest -v tests/testnet/aio/*

## 2.7 BitShares classes inheritance

This document briefly describes how bitshares.xxx classes are inherited

### 2.7.1 AbstractBlockchainInstanceProvider (graphenelib) role

Typical class inheritance is *Foo(GrapheneFoo) -> GrapheneFoo(AbstractBlockchainInstanceProvider) -> AbstractBlockchainInstanceProvider*

This class provides access to RPC via *self.blockchain* property, which is set to *blockchain\_instance* kwarg or *shared\_blockchain\_instance* as a fallback. *shared\_blockchain\_instance* in turn gets proper blockchain instance class calling *self.get\_instance\_class()*. *get\_instance\_class()* is overwritten in *bitshares.instance.BlockchainInstance*

*inject* method (used as *@BlockchainInstance.inject* decorator) is needed to provide blockchain instance class in common manner.

In short, *Foo + @BlockchainInstance.inject* init calls *AbstractBlockchainInstanceProvider.\_\_init\_\_* and *Foo.\_\_init\_\_*.

*AbstractBlockchainInstanceProvider.\_\_init\_\_* sets *self.\_blockchain* from kwarg or via calling *self.shared\_blockchain\_instance()*, which leads to initializing *bitshares.Bitshares* (*bitshares.Bitshares* is inherited from *AbstractGrapheneChain*).

### 2.7.2 Asyncio versions

Typical async class inherited from corresponding async class from graphenelib, and from synchronous class, like *class Asset(GrapheneAsset, SyncAsset)*. So, async version needs to redefine only needed methods.

Most of async classes needs async *\_\_init\_\_* because they're loading some objects from the blockchain, which requires an API call performed via async RPC. To achieve this, async *AbstractBlockchainInstanceProvider* has different *inject()* method.

## 2.8 Contributing

# Contributing

## Introduction

You are here to help BitShares and the python-bitshares library? Awesome, feel welcome and read the following sections in order to know how to ask questions and how to work on something.

## Code of Conduct

All members of our community are expected to follow our [Code of Conduct](CODE\_OF\_CONDUCT.md). Please make sure you are welcoming and friendly in all of our spaces.

## Get in touch

Feel free to get in contact with the developer community on [Telegram](https://t.me/pybitshares).

## Contributing to development

When contributing to this repository, please first discuss the change you wish to make via issue, email, or any other method with the owners of this repository before making a change.

Please note we have a code of conduct, please follow it in all your interactions with the project.

### Your First Contribution

To familiarize yourself with the code and procedures, we recommend to get started with:

- review a Pull Request
- fix an Issue
- update the documentation

- make a website
- write a tutorial

### ### Git Flow

This project makes heavy use of [git flow](<http://nvie.com/posts/a-successful-git-branching-model/>). If you are not familiar with it, then the most important thing for you to understand is that:

**pull requests need to be made against the ‘develop’ branch!**

### ### Conventional Commits

We strictly use [conventional commits](<https://www.conventionalcommits.org/en/v1.0.0/#summary>)!

### ### Contributor License Agreement

Upon submission of a pull request, you will be asked to sign the [Contributor License Agreement](CLA.md) digitally through a github-connected service.

### ### 1. Where do I go from here?

If you’ve noticed a bug or have a question, [search the issue tracker](<https://github.com/bitshares/python-bitshares/issues>) to see if someone else in the community has already created a ticket. If not, go ahead and [make one](<https://github.com/bitshares/python-bitshares/issues/new>)!

### ### 2. Fork & create a branch

If this is something you think you can fix, then fork the repository and create a branch with a descriptive name.

A good branch name would be (where issue #325 is the ticket you’re working on):

```
git checkout -b 325-new-fancy-feature
```

### ### 3. Get the test suite running

Make sure to add a unit tests for your code contribution in *tests/*. You may use other unit tests as template for your own tests.

Individual unit tests can be run by:

```
python3 -m unittest tests/test_NEWFEATURE.py
```

The entire test suite can be run in a sandbox through

```
tox
```

### ### 4. Did you find a bug?

- **Ensure the bug was not already reported** by [searching all issues](<https://github.com/bitshares/python-bitshares/issues>).
- If you’re unable to find an open issue addressing the problem, [open a new one](<https://github.com/bitshares/python-bitshares/issues/new>). Be sure to include a **title and clear description**, as much relevant information as possible, and a **code sample** or an **executable test case** demonstrating the expected behavior that is not occurring.
- If possible, use the relevant bug report templates to create the issue. Simply copy the content of the appropriate template into a .py file, make the necessary changes to demonstrate the issue, and **paste the content into the issue description**.

### ### 5. Implement your fix or feature

At this point, you’re ready to make your changes! Feel free to ask for help; everyone is a beginner at first

### ### 6. Get the style right



Your patch should follow the same conventions & pass the same code quality checks as the rest of the project. [Codeclimate](<https://codeclimate.com/github/bitshares/python-bitshares>) will give you feedback in this regard. You can check & fix codeclimate's feedback by running it locally using [Codeclimate's CLI](<https://codeclimate.com/github/bitshares/python-bitshares>), via *codeclimate analyze*.

### ### 7. Make a Pull Request

#### **Pull requests are supposed to go against the 'develop' branch, only!**

At this point, you should switch back to your *develop* branch and make sure it's up to date with python-bitshares's *develop* branch:

```
git remote add upstream git@github.com:bitshares/python-bitshares.git
git checkout develop
git pull upstream develop
```

Then update your feature branch from your local copy of develop, and push it!

```
git checkout 325-new-fancy-feature
git rebase develop
git push --set-upstream origin 325-new-fancy-feature
```

Finally, go to GitHub and make a Pull Request :D

Travis CI will run our test suite against all supported Rails versions. We care about quality, so your PR won't be merged until all tests pass. It's unlikely, but it's possible that your changes pass tests in one Rails version but fail in another. In that case, you'll have to setup your development environment (as explained in step 3) to use the problematic Rails version, and investigate what's going on!

### ### 7. Keeping your Pull Request updated

If a maintainer asks you to "rebase" your PR, they're saying that a lot of code has changed, and that you need to update your branch so it's easier to merge.

To learn more about rebasing in Git, there are a lot of good git rebasing resources but here's the suggested workflow:

```
git checkout 325-new-fancy-feature
git pull --rebase upstream develop
git push --force-with-lease 325-new-fancy-feature
```

### ### 8. Merging a PR (maintainers only)

A PR can only be merged into *develop* by a maintainer if:

- Pull request goes against *develop* branch.
- It is passing CI.
- It has been approved by at least two maintainers. If it was a maintainer who opened the PR, only one extra approval is needed.
- It has no requested changes.
- It is up to date with current *develop*.
- Did the contributor sign the [CLA](CLA.md)

Any maintainer is allowed to merge a PR if all of these conditions are met.

### ### 9. Shipping a release (maintainers only)

Create a new pull request with *release/<YYYYmmdd>*. Push the branch. When merged into master, release tagging and publishing will be triggered through github actions.

## 2.9 Support and Questions

- <https://t.me/pybitshares> - dedicated channel

- <https://bitsharestalk.org> - BitShares forum
- <https://t.me/BitSharesDEX> - general telegram channel

---

### Note:

All methods that construct and sign a transaction can be given the `account=` parameter to identify the user that is going to be affected by this transaction, e.g.:

- the source account in a transfer
- the account that buys/sells an asset in the exchange
- the account whose collateral will be modified

**Important,** If no `account` is given, then the `default_account` according to the settings in `config` is used instead.

---

## 3.1 Create a wallet

```
from bitshares import BitShares
bitshares = BitShares()
bitshares.wallet.create("secret-passphrase")
bitshares.wallet.addPrivateKey("<wif-key>")
```

## 3.2 Unlock the wallet for a transfer

```
from bitshares import BitShares
bitshares = BitShares()
bitshares.wallet.unlock("wallet-passphrase")
bitshares.transfer("<to>", "<amount>", "<asset>", "[<memo>]", account="<from>")
```

### 3.3 Monitor the BitShares Blockchain operation-wise

```
from bitshares.blockchain import Blockchain
blockchain = Blockchain()
for op in Blockchain.ops():
    print(op)
```

### 3.4 Obtain the content of one block

```
from bitshares.block import Block
print(Block(1))
```

### 3.5 Obtain Account balance, open orders and history

```
from bitshares.account import Account
account = Account("init0")
print(account.balances)
print(account.openorders)
for h in account.history():
    print(h)
```

### 3.6 Print Market ticker and sell

```
from bitshares.market import Market
market = Market("USD:BTS")
print(market.ticker())
market.bitshares.wallet.unlock("wallet-passphrase")
print(market.sell(300, 100)) # sell 100 USD for 300 BTS/USD
```

### 3.7 Adjust collateral

```
from bitshares.dex import Dex
dex = Dex()
dex.bitshares.wallet.unlock("wallet-passphrase")
dex.adjust_collateral_ratio("SILVER", 3.5)
```

## CHAPTER 4

---

### Developers and Community

---

Discussions around development and use of this library can be found in a [dedicated Telegram Channel](<https://t.me/pybitshares>)



## 5.1 bitshares

### 5.1.1 bitshares package

#### Subpackages

#### bitshares.aio package

#### Submodules

#### bitshares.aio.account module

```
class bitshares.aio.account.Account (*args, **kwargs)
    Bases: bitshares.aio.instance.BlockchainInstance, bitshares.aio.account.Account
```

This class allows to easily access Account data.

#### Parameters

- **account\_name** (*str*) – Name of the account
- **blockchain\_instance** (*bitshares.aio.bitshares.BitShares*) – BitShares instance
- **full** (*bool*) – Obtain all account data including orders, positions, etc.
- **lazy** (*bool*) – Use lazy loading
- **full** – Obtain all account data including orders, positions, etc.

**Returns** Account data

**Return type** dictionary

Raises `bitshares.exceptions.AccountDoesNotExistException` – if account does not exist

Instances of this class are dictionaries that come with additional methods (see below) that allow dealing with an account and it's corresponding functions.

```
from bitshares.aio.account import Account
account = await Account("init0")
print(account)
```

---

**Note:** This class comes with its own caching function to reduce the load on the API server. Instances of this class can be refreshed with `await Account.refresh()`.

---

**balance** (*symbol*)

Obtain the balance of a specific Asset. This call returns instances of `amount.Amount`.

**balances**

List balances of an account. This call returns instances of `amount.Amount`.

**bitshares**

Alias for the specific blockchain.

**blacklist** (*account*)

Add an other account to the blacklist of this account

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.aio.instance.BlockchainInstance`

**classmethod cache\_object** (*data, key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**call\_positions**

Alias for `:func:bitshares.account.Account.callpositions`.

**callpositions**

List call positions (collateralized positions *Market Pegged Assets*)

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**ensure\_full** ()

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. `bitshares.BitShares`



**getfromcache** (*id*)

Get an element from the cache explicitly

**history** (*first=0, last=0, limit=-1, only\_ops=[], exclude\_ops=[]*)

Returns a generator for individual account transactions. The latest operation will be first. This call can be used in a `for` loop.

#### Parameters

- **first** (*int*) – sequence number of the first transaction to return (*optional*)
- **last** (*int*) – sequence number of the last transaction to return (*optional*)
- **limit** (*int*) – limit number of transactions to return (*optional*)
- **only\_ops** (*array*) – Limit generator by these operations (*optional*)
- **exclude\_ops** (*array*) – Exclude these operations from generator (*optional*).

... **note::** `only_ops` and `exclude_ops` takes an array of strings: The full list of operation ID's can be found in `operationids.py`. Example: `['transfer', 'fill_order']`

**identifier** = `None`

**incached** (*id*)

Is an element cached?

**classmethod inject** (*cls*)

**is\_fully\_loaded**

Is this instance fully loaded / e.g. all data available?

**is\_ltm**

Is the account a lifetime member (LTM)?

**items** ()

This overrides `items()` so that `refresh()` is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**name**

**nolist** (*account*)

Remove an other account from any list of this account

**static objectid\_valid** (*i*)

Test if a string looks like a regular object id of the form::

```
xxxx.yyzz.zzzz
```

with those being numbers.

**openorders**

Returns open Orders.

**perform\_id\_tests** = `True`

**pop** (*k*, *d*) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a

2-tuple; but raise `KeyError` if *D* is empty.

**refresh** ()

Refresh/Obtain an account's data from the API server

**static** `set_cache_store (klass, *args, **kwargs)`

**classmethod** `set_shared_blockchain_instance (instance)`

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** `instance (chaininstance)` – Chain instance

**classmethod** `set_shared_config (config)`

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance ()**

This method allows to set the current instance as default

**setdefault ()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance ()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id = 1**

**store (data, key='id')**

Cache the list

**Parameters** `data (list)` – List of objects to cache

**test\_valid\_objectid (i)**

Alias for `objectid_valid`

**testid (id)**

In contrast to validity, this method tests if the `objectid` matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id = None**

**type\_ids = []**

**update ([E], \*\*F) → None.** Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**upgrade ()**

Upgrade account to life time member

**values () →** an object providing a view on D's values

**whitelist (account)**

Add an other account to the whitelist of this account

**class** `bitshares.aio.account.AccountUpdate (*args, **kwargs)`

Bases: `bitshares.aio.instance.BlockchainInstance`, `bitshares.aio.account.AccountUpdate`

This purpose of this class is to keep track of account updates as they are pushed through by `bitshares.notify.Notify`.

Instances of this class are dictionaries and take the following form:

**account**

In order to obtain the actual `account.Account` from this class, you can use the `account` attribute.

**bitshares**

Alias for the specific blockchain.

**blockchain****blockchain\_instance\_class**

alias of *bitshares.aio.instance.BlockchainInstance*

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject** (cls)

**items** () → a set-like object providing a view on D's items

**keys** () → a set-like object providing a view on D's keys

**pop** (k[, d]) → v, remove specified key and return the corresponding value.  
If key is not found, d is returned if given, otherwise *KeyError* is raised

**popitem** () → (k, v), remove and return some (key, value) pair as a  
2-tuple; but raise *KeyError* if D is empty.

**classmethod set\_shared\_blockchain\_instance** (instance)

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters** *instance* (chaininstance) – Chain instance

**classmethod set\_shared\_config** (config)

This allows to set a config that will be used when calling *shared\_blockchain\_instance* and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize *SharedInstance.instance* and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**update** ([E], \*\*F) → None. Update D from dict/iterable E and F.

If E is present and has a *.keys()* method, then does: for k in E: D[k] = E[k] If E is present and lacks a *.keys()* method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

## bitshares.aito.amount module

**class** bitshares.aito.amount.**Amount** (\*args, \*\*kwargs)

Bases: *bitshares.aito.instance.BlockchainInstance, bitshares.aito.amount.Amount*

This class deals with Amounts of any asset to simplify dealing with the tuple:

`(amount, asset)`

### Parameters

- **args** (*list*) – Allows to deal with different representations of an amount
- **amount** (*float*) – Let's create an instance with a specific amount
- **asset** (*str*) – Let's you create an instance with a specific asset (symbol)
- **blockchain\_instance** (*bitshares.aito.bitshares.BitShares*) – Bit-Shares instance

**Returns** All data required to represent an Amount/Asset

**Return type** dict

**Raises** **ValueError** – if the data provided is not recognized

```
from bitshares.aito.amount import Amount
from bitshares.aito.asset import Asset
a = await Amount("1 USD")
b = await Amount(1, "USD")
c = await Amount("20", await Asset("USD"))
```

Way to obtain a proper instance:

- args can be a string, e.g.: "1 USD"
- args can be a dictionary containing amount and asset\_id
- args can be a dictionary containing amount and asset
- args can be a list of a float and str (symbol)
- args can be a list of a float and a *bitshares.aito.asset.Asset*
- amount and asset are defined manually

An instance is a dictionary and comes with the following keys:

- amount (float)
- symbol (str)
- asset (instance of *bitshares.aito.asset.Asset*)

### amount

Returns the amount as float

### asset

Returns the asset as instance of *asset.Asset*

### bitshares

Alias for the specific blockchain.

### blockchain

**blockchain\_instance\_class**alias of *bitshares.aio.instance.BlockchainInstance***chain**

Short form for blockchain (for the lazy)

**clear()** → None. Remove all items from D.**copy()**

Copy the instance and make sure not to use a reference

**define\_classes()**

Needs to define instance variables that provide classes

**fromkeys()**

Create a new dictionary with keys from iterable and values set to value.

**get()**

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class()**Should return the Chain instance class, e.g. *bitshares.BitShares***classmethod inject(cls)****items()** → a set-like object providing a view on D's items**json()****keys()** → a set-like object providing a view on D's keys**pop(k[, d])** → v, remove specified key and return the corresponding value.If key is not found, d is returned if given, otherwise *KeyError* is raised**popitem()** → (k, v), remove and return some (key, value) pair as a2-tuple; but raise *KeyError* if D is empty.**classmethod set\_shared\_blockchain\_instance(instance)**This method allows us to override default instance for all users of *SharedInstance.instance*.**Parameters instance(chaininstance)** – Chain instance**classmethod set\_shared\_config(config)**This allows to set a config that will be used when calling *shared\_blockchain\_instance* and allows to define the configuration without requiring to actually create an instance**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**This method will initialize *SharedInstance.instance* and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.**symbol**

Returns the symbol of the asset

**tuple()**

**update** (*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

## bitshares.aio.asset module

**class** `bitshares.aio.asset.Asset` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.aio.instance.BlockchainInstance`, `bitshares.aio.asset.Asset`

BitShares asset.

Async version of `bitshares.bitshares.Asset`

**add\_authorities** (*type*, *authorities=None*)

Add authorities to an assets white/black list.

### Parameters

- **type** (*str*) – blacklist or whitelist
- **authorities** (*list*) – List of authorities (Accounts)

**add\_markets** (*type*, *authorities=None*, *force\_enable=True*)

Add markets to an assets white/black list.

### Parameters

- **type** (*str*) – blacklist or whitelist
- **markets** (*list*) – List of markets (assets)
- **force\_enable** (*bool*) – Force enable `white_list` flag

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.aio.instance.BlockchainInstance`

**classmethod** `cache_object` (*data*, *key=None*)

This classmethod allows to feed an object into the cache is mostly used for testing

**calls**

**chain**

Short form for blockchain (for the lazy)

**change\_issuer** (*new\_issuer*, *\*\*kwargs*)

Change asset issuer (needs signing with owner key!)

**Parameters** **new\_issuer** (*str*) – account name

**clear** () → None. Remove all items from D.

**classmethod** `clear_cache` ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**disableflag** (*flag*)

Enable a certain flag.

**Parameters** **flag** (*str*) – Flag name

**enableflag** (*flag*)

Enable a certain flag.

**Parameters** **flag** (*str*) – Flag name

**ensure\_full** ()

**feed**

**feeds**

**flags**

List the permissions that are currently used (flags)

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_call\_orders** (*limit=100*)

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**get\_settle\_orders** (*limit=100*)

**getfromcache** (*id*)

Get an element from the cache explicitly

**halt** ()

Halt this asset from being moved or traded.

**identifier** = **None**

**incached** (*id*)

Is an element cached?

**classmethod inject** (*cls*)

**is\_bitasset**

Is the asset a market pegged asset?

**is\_fully\_loaded**

Is this instance fully loaded / e.g. all data available?

**issue** (*amount, to, memo=None, \*\*kwargs*)

Issue new shares of an asset.

**Parameters**

- **amount** (*float*) – Amount to issue
- **to** (*str*) – Recipient
- **memo** (*str*) – (optional) Memo message

**items** ()

This overrides items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**market\_fee\_percent**

**max\_market\_fee**

**static objectid\_valid**(*i*)

Test if a string looks like a regular object id of the form::

`xxxx.yyyyy.zzzz`

with those being numbers.

**perform\_id\_tests** = **True**

**permissions**

List the permissions for this asset that the issuer can obtain

**pop**(*k*[, *d*]) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise **KeyError** is raised

**popitem**() → (*k*, *v*), remove and return some (key, value) pair as a

2-tuple; but raise **KeyError** if *D* is empty.

**precision**

**refresh**()

Refresh the data from the API server

**release**(*whitelist\_authorities=None*, *blacklist\_authorities=None*, *whitelist\_markets=None*, *blacklist\_markets=None*)

Release this asset and allow unrestricted transfer, trading, etc.

#### Parameters

- **whitelist\_authorities** (*list*) – List of accounts that serve as whitelist authorities
- **blacklist\_authorities** (*list*) – List of accounts that serve as blacklist authorities
- **whitelist\_markets** (*list*) – List of assets to allow trading with
- **blacklist\_markets** (*list*) – List of assets to prevent trading with

**remove\_authorities**(*type*, *authorities=None*)

Remove authorities from an assets white/black list.

#### Parameters

- **type** (*str*) – blacklist or whitelist
- **authorities** (*list*) – List of authorities (Accounts)

**remove\_markets**(*type*, *authorities=None*)

Remove markets from an assets white/black list.

#### Parameters

- **type** (*str*) – blacklist or whitelist
- **markets** (*list*) – List of markets (assets)

**seize**(*\*args*)

Seize amount from an account and send to another.

... **note::** This requires the **override\_authority** to be set for this asset!



**Parameters**

- **from\_account** (`bitshares.account.Account`) – From this account
- **to\_account** (`bitshares.account.Account`) – To this account
- **amount** (`bitshares.amount.Amount`) – Amount to seize

**static set\_cache\_store** (*klass, \*args, \*\*kwargs*)

**set\_market\_fee** (*percentage\_fee, max\_market\_fee*)

Set trading percentage fee.

**Parameters**

- **percentage\_fee** (*float*) – Percentage of fee
- **max\_market\_fee** (`bitshares.amount.Amount`) – Max Fee

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** **instance** (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**setoptions** (*flags*)

Enable a certain flag.

Flags:

- `charge_market_fee`
- `white_list`
- `override_authority`
- `transfer_restricted`
- `disable_force_settle`
- `global_settle`
- `disable_confidential`
- `witness_fed_asset`
- `committee_fed_asset`

**Parameters** **flag** (*dict*) – dictionary of flags and boolean

**settlements**

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (*data*, *key*='id')  
Cache the list

**Parameters** **data** (*list*) – List of objects to cache

**symbol**

**test\_valid\_objectid** (*i*)  
Alias for objectid\_valid

**testid** (*id*)  
In contrast to validity, this method tests if the objectid matches the type\_id provided in self.type\_id or self.type\_ids

**type\_id** = None

**type\_ids** = []

**update** ([*E*], **\*\*F**) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**update\_cer** (*cer*, *account*=None, **\*\*kwargs**)  
Update the Core Exchange Rate (CER) of an asset

**update\_feed\_producers** (*producers*)  
Update bitasset feed producers.

**Parameters** **producers** (*list*) – List of accounts that are allowed to produce a feed

**values** () → an object providing a view on D's values

## bitshares.aio.bitshares module

**class** bitshares.aio.bitshares.**BitShares** (\*args, **\*\*kwargs**)  
Bases: graphenecommon.aio.chain.AbstractGrapheneChain, [bitshares.bitshares.BitShares](#)

BitShares async client.

This is an asyncio version of bitshares.BitShares

**Parameters** **loop** (*object*) – asyncio event loop

Example usage:

```
bitshares = BitShares(loop=loop)
await bitshares.connect()
```

**account\_whitelist** (*account\_to\_whitelist*, *lists*=None, *account*=None, **\*\*kwargs**)  
Account whitelisting.

**Parameters**

- **account\_to\_whitelist** (*str*) – The account we want to add to either the white- or the blacklist
- **lists** (*set*) – (defaults to ('white')). Lists the user should be added to. Either empty set, 'black', 'white' or both.

- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**allow** (*foreign*, *weight=None*, *permission='active'*, *account=None*, *threshold=None*, *\*\*kwargs*)

Give additional access to an account by some other public key or account.

#### Parameters

- **foreign** (*str*) – The foreign account that will obtain access
- **weight** (*int*) – (optional) The weight to use. If not define, the threshold will be used. If the weight is smaller than the threshold, additional signatures will be required. (defaults to `threshold`)
- **permission** (*str*) – (optional) The actual permission to modify (defaults to `active`)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)
- **threshold** (*int*) – The threshold that needs to be reached by signatures to be able to interact

**approvecommittee** (*committees*, *account=None*, *\*\*kwargs*)

Approve a committee.

#### Parameters

- **committees** (*list*) – list of committee member name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**approveproposal** (*proposal\_ids*, *account=None*, *approver=None*, *\*\*kwargs*)

Approve Proposal.

#### Parameters

- **proposal\_id** (*list*) – Ids of the proposals
- **approver** (*str*) – The account or key to use for approval (defaults to `account`)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**approvewitness** (*witnesses*, *account=None*, *\*\*kwargs*)

Approve a witness.

#### Parameters

- **witnesses** (*list*) – list of Witness name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**approveworker** (*workers*, *account=None*, *\*\*kwargs*)

Approve a worker.

#### Parameters

- **workers** (*list*) – list of worker member name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**asset\_settle** (*amount*, *account=None*, *\*\*kwargs*)

**bid\_collateral** (*additional\_collateral*, *debt\_covered*, *account=None*, *\*\*kwargs*)

**broadcast** (*tx=None*)

Broadcast a transaction to the Blockchain

**Parameters** **tx** (*tx*) – Signed transaction to broadcast

**cancel** (*orderNumbers, account=None, \*\*kwargs*)

Cancels an order you have placed in a given market. Requires only the “orderNumbers”. An order number takes the form 1.7.xxx.

**Parameters** **orderNumbers** (*str*) – The Order Object ide of the form 1.7.xxxx

**cancel\_subscriptions** ()

Cancel all active subscriptions.

**clear** ()

**clear\_cache** ()

Clear Caches

**connect** ()

Connect to blockchain network

Async version does wallet initialization after connect because wallet depends on prefix which is available after connection only, and we want to keep `__init__()` synchronous.

**create\_account** (*account\_name, registrar=None, referrer='1.2.35641', referrer\_percent=50, owner\_key=None, active\_key=None, memo\_key=None, owner\_account=None, active\_account=None, password=None, additional\_owner\_keys=None, additional\_active\_keys=None, additional\_owner\_accounts=None, additional\_active\_accounts=None, proxy\_account='proxy-to-self', storekeys=True, \*\*kwargs*)

Create new account on BitShares.

The brainkey/password can be used to recover all generated keys (see *bitsharesbase.account* for more details.

By default, this call will use `default_account` to register a new name `account_name` with all keys being derived from a new brain key that will be returned. The corresponding keys will automatically be installed in the wallet.

**Warning:** Don't call this method unless you know what you are doing! Be sure to understand what this method does and where to find the private keys for your account.

---

**Note:** Please note that this imports private keys (if password is present) into the wallet by default. However, it **does not import the owner key** for security reasons. Do NOT expect to be able to recover it from the wallet if you lose your password!

---

#### Parameters

- **account\_name** (*str*) – (**required**) new account name
- **registrar** (*str*) – which account should pay the registration fee (defaults to `default_account`)
- **owner\_key** (*str*) – Main owner key
- **active\_key** (*str*) – Main active key
- **memo\_key** (*str*) – Main memo\_key

- **password** (*str*) – Alternatively to providing keys, one can provide a password from which the keys will be derived
- **additional\_owner\_keys** (*array*) – Additional owner public keys
- **additional\_active\_keys** (*array*) – Additional active public keys
- **additional\_owner\_accounts** (*array*) – Additional owner account names
- **additional\_active\_accounts** (*array*) – Additional active account names
- **storekeys** (*bool*) – Store new keys in the wallet (default: `True`)

Raises **`AccountExistsException`** – if the account already exists on the blockchain

**create\_asset** (*symbol*, *precision*, *max\_supply*, *description*=", *is\_bitasset*=*False*, *is\_prediction\_market*=*False*, *market\_fee\_percent*=0, *max\_market\_fee*=*None*, *permissions*=*None*, *flags*=*None*, *whitelist\_authorities*=*None*, *blacklist\_authorities*=*None*, *whitelist\_markets*=*None*, *blacklist\_markets*=*None*, *bitasset\_options*=*None*, *account*=*None*, *\*\*kwargs*)

Create a new asset.

#### Parameters

- **symbol** (*str*) – Asset symbol
- **precision** (*int*) – Asset precision
- **max\_supply** (*int*) – Asset max supply
- **description** (*str*) – (optional) Asset description
- **is\_bitasset** (*bool*) – (optional) `True` = bitasset, `False` = UIA (default: `False`)
- **is\_prediction\_market** (*bool*) – (optional) `True`: PD, `False` = plain smartcoin (default: `False`)
- **market\_fee\_percent** (*float*) – (optional) Charge market fee (0-100) (default: 0)
- **max\_market\_fee** (*float*) – (optional) Absolute amount of max market fee, value of this option should be a whole number (default: same as `max_supply`)
- **permissions** (*dict*) – (optional) Asset permissions
- **flags** (*dict*) – (optional) Enabled asset flags
- **whitelist\_authorities** (*list*) – (optional) List of accounts that serve as whitelist authorities
- **blacklist\_authorities** (*list*) – (optional) List of accounts that serve as blacklist authorities
- **whitelist\_markets** (*list*) – (optional) List of assets to allow trading with
- **blacklist\_markets** (*list*) – (optional) List of assets to prevent trading with
- **bitasset\_options** (*dict*) – (optional) Bitasset settings
- **account** (*str*) – (optional) the issuer account to (defaults to `default_account`)

**create\_committee\_member** (*url*=", *account*=*None*, *\*\*kwargs*)

Create a committee member.

#### Parameters

- **url** (*str*) – URL to read more about the worker

- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**create\_liquidity\_pool** (*asset\_a*, *asset\_b*, *share\_asset*, *taker\_fee\_percent*, *withdrawal\_fee\_percent*, *account=None*, *\*\*kwargs*)

Create a liquidity pool.

#### Parameters

- **asset\_a** (*str*) – First asset in the pool pair.
- **asset\_b** (*str*) – Second asset in the pool pair.
- **share\_asset** (*str*) – The asset which represents shares in the pool.

For asset parameters, these can be either symbols or `asset_id` strings. Note that network expects `asset_a` to have a lower-numbered `asset_id` than `asset_b`.

#### Parameters

- **taker\_fee\_percent** (*float*) – The pool’s taker fee percentage.
- **withdrawal\_fee\_percent** (*float*) – The pool’s withdrawal fee percent.

For percentages, meaningful range is [0.00, 100.00], where 1% is represented as 1.0. Smallest non-zero value recognized by BitShares chain is 0.01 for 0.01%.

**create\_voting\_ticket** (*target\_type*, *amount\_to\_lock*, *account=None*, *\*\*kwargs*)

Create a voting ticket.

#### Parameters

- **target\_type** (*int*, *str*) – Lock period target. Should be a string from `operations.ticket_type_strings` or the index of the intended string.
- **amount\_to\_lock** (*Amount*) – Amount to lock up for the duration selected in `target_type`.

**create\_worker** (*name*, *daily\_pay*, *end*, *url=""*, *begin=None*, *payment\_type='vesting'*, *pay\_vesting\_period\_days=0*, *account=None*, *\*\*kwargs*)

Create a worker.

This removes the shares from the supply

### Required

#### Parameters

- **name** (*str*) – Name of the worker
- **daily\_pay** (*bitshares.amount.Amount*) – The amount to be paid daily
- **end** (*datetime*) – Date/time of end of the worker

### Optional

#### Parameters

- **url** (*str*) – URL to read more about the worker
- **begin** (*datetime*) – Date/time of begin of the worker
- **payment\_type** (*string*) – [“burn”, “refund”, “vesting”] (default: “vesting”)
- **pay\_vesting\_period\_days** (*int*) – Days of vesting (default: 0)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**define\_classes** ()

**delete\_liquidity\_pool** (*pool*, *account=None*, *\*\*kwargs*)

Delete a liquidity pool.

**Parameters** *pool* (*str*, *Asset*) – The liquidity pool to delete. Can be the pool id as a string, or can be an Asset, *asset\_id*, or symbol of the share asset for the pool.

**deposit\_into\_liquidity\_pool** (*pool*, *amount\_a*, *amount\_b*, *account=None*, *\*\*kwargs*)

Deposit assets into a liquidity pool.

**Parameters**

- **pool** (*str*, *Asset*) – The liquidity pool to use. Can be the pool id as a string, or can be an Asset, *asset\_id*, or symbol of the share asset for the pool.
- **amount\_a** (*Amount*) –
- **amount\_b** (*Amount*) –

**disallow** (*foreign*, *permission='active'*, *account=None*, *threshold=None*, *\*\*kwargs*)

Remove additional access to an account by some other public key or account.

**Parameters**

- **foreign** (*str*) – The foreign account that will obtain access
- **permission** (*str*) – (optional) The actual permission to modify (defaults to *active*)
- **account** (*str*) – (optional) the account to allow access to (defaults to *default\_account*)
- **threshold** (*int*) – The threshold that needs to be reached by signatures to be able to interact

**disapprovecommittee** (*committees*, *account=None*, *\*\*kwargs*)

Disapprove a committee.

**Parameters**

- **committees** (*list*) – list of committee name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to *default\_account*)

**disapproveproposal** (*proposal\_ids*, *account=None*, *approver=None*, *\*\*kwargs*)

Disapprove Proposal.

**Parameters**

- **proposal\_ids** (*list*) – Ids of the proposals
- **account** (*str*) – (optional) the account to allow access to (defaults to *default\_account*)

**disapprovewitness** (*witnesses*, *account=None*, *\*\*kwargs*)

Disapprove a witness.

**Parameters**

- **witnesses** (*list*) – list of Witness name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to *default\_account*)

**disapproveworker** (*workers*, *account=None*, *\*\*kwargs*)

Disapprove a worker.

**Parameters**

- **workers** (*list*) – list of worker name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**exchange\_with\_liquidity\_pool** (*pool*, *amount\_to\_sell*, *min\_to\_receive*, *account=None*, *\*\*kwargs*)

Exchange assets against a liquidity pool.

**Parameters**

- **pool** (*str*, *Asset*) – The liquidity pool to use. Can be the pool id as a string, or can be an *Asset*, *asset\_id*, or symbol of the share asset for the pool.
- **amount\_to\_sell** (*Amount*) –
- **min\_to\_receive** (*Amount*) –

**finalizeOp** (*ops*, *account*, *permission*, *\*\*kwargs*)

This method obtains the required private keys if present in the wallet, finalizes the transaction, signs it and broadcasts it

**Parameters**

- **ops** (*operation*) – The operation (or list of operations) to broadcast
- **account** (*operation*) – The account that authorizes the operation
- **permission** (*string*) – The required permission for signing (active, owner, posting)
- **append\_to** (*object*) – This allows to provide an instance of *ProposalsBuilder* (see [\*new\\_proposal\(\)\*](#)) or *TransactionBuilder* (see [\*new\\_tx\(\)\*](#)) to specify where to put a specific operation.

... note:: **append\_to** is exposed to every method used in the this class

... note:

If ``ops`` is a list of operation, they all need to be signable by the same key! Thus, you cannot combine ops that require active permission with ops that require posting permission. Neither can you use different accounts for different operations!

... note:: This uses **txbuffer** as instance of `transactionbuilder.TransactionBuilder`. You may want to use your own txbuffer

**fund\_fee\_pool** (*symbol*, *amount*, *account=None*, *\*\*kwargs*)

Fund the fee pool of an asset.

**Parameters**

- **symbol** (*str*) – The symbol to fund the fee pool of
- **amount** (*float*) – The amount to be burned.
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)



**htlc\_create** (*amount*, *to*, *preimage*, *hash\_type*='ripemd160', *account*=None, *expiration*=3600, *\*\*kwargs*)

Create an HTLC contract.

#### Parameters

- **amount** (*Amount*) – Amount to lock
- **to** (*str*) – Recipient
- **expiration** (*int*) – Contract duration in seconds
- **hash\_hex** (*str*) – Hash as string of hex digits
- **preimage** (*str*) – Preimage as ascii string. Note hex digits would be interpreted as ascii text, not as bytes. Not generally recommended to use this option. Options `hash_hex` and `preimage` are mutually exclusive.
- **preimage\_length** (*int*) – If non-zero, htlc contract will require preimage of exact length. Generally OK to leave this as zero. Note if preimage param is provided, this value SHOULD be either zero or match exactly the length of the preimage, else an irredeemable htlc will be created. Optionally, a sentinel value of -1 can be used to compute length automatically from the preimage param.

**htlc\_redeem** (*htlc\_id*, *preimage*, *account*=None, *\*\*kwargs*)

Redeem an htlc contract.

#### Parameters

- **preimage** (*str*) – The preimage that unlocks the htlc
- **encoding** (*str*) – “utf-8”, ..., or “hex”

**info** ()

Returns the global properties

**is\_connected** ()

**newWallet** (*pwd*)

**new\_proposal** (*parent*=None, *proposer*=None, *proposal\_expiration*=None, *proposal\_review*=None, *\*\*kwargs*)

**new\_tx** (*\*args*, *\*\*kwargs*)

Let's obtain a new txbuffer

**Returns int txid** id of the new txbuffer

**new\_wallet** (*pwd*)

Create a new wallet. This method is basically only calls `wallet.Wallet.create()`.

**Parameters** *pwd* (*str*) – Password to use for the new wallet

**Raises** `exceptions.WalletExists` – if there is already a wallet created

**prefix**

Contains the prefix of the blockchain

**propbuffer**

Return the default proposal buffer

**proposal** (*proposer*=None, *proposal\_expiration*=None, *proposal\_review*=None)

Return the default proposal buffer

... **note::** If any parameter is set, the default proposal parameters will be changed!

**publish\_price\_feed** (*symbol, settlement\_price, cer=None, mssr=110, mcr=200, account=None*)

Publish a price feed for a market-pegged asset.

#### Parameters

- **symbol** (*str*) – Symbol of the asset to publish feed for
- **settlement\_price** (`bitshares.price.Price`) – Price for settlement
- **cer** (`bitshares.price.Price`) – Core exchange Rate (default `settlement_price + 5%`)
- **mssr** (*float*) – Percentage for max short squeeze ratio (default: 110%)
- **mcr** (*float*) – Percentage for maintenance collateral ratio (default: 200%)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

---

**Note:** The `account` needs to be allowed to produce a price feed for `symbol`. For witness produced feeds this means `account` is a witness account!

---

**registrar = None**

Generate new keys from password

**reserve** (*amount, account=None, \*\*kwargs*)

Reserve/Burn an amount of this shares.

This removes the shares from the supply

#### Parameters

- **amount** (`bitshares.amount.Amount`) – The amount to be burned.
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**set\_blocking** (*block=True*)

This sets a flag that forces the broadcast to block until the transactions made it into a block

**set\_default\_account** (*account*)

Set the default account to be used

**set\_proxy** (*proxy\_account, account=None, \*\*kwargs*)

Set a specific proxy for account.

#### Parameters

- **proxy\_account** (`bitshares.account.Account`) – Account to be proxied
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**set\_shared\_instance** ()

This method allows to set the current instance as default

**sign** (*tx=None, wifs=[]*)

Sign a provided transaction with the provided key(s)

#### Parameters

- **tx** (*dict*) – The transaction to be signed and returned

- **wifs** (*string*) – One or many wif keys to use for signing a transaction. If not present, the keys will be loaded from the wallet as defined in “missing\_signatures” key of the transactions.

**subscribe\_to\_accounts** (*accounts, event\_id=1*)

Activate subscription to account-related events.

**Parameters**

- **accounts** (*list*) – account names or ids to subscribe
- **event\_id** (*int*) – id of this subscription in upcoming notifications

**subscribe\_to\_blocks** (*event\_id=2*)

Activate subscription to block.

Each time block is applied an event will occur in self.notifications.

**Parameters** **event\_id** (*int*) – id of this subscription in upcoming notifications

**subscribe\_to\_market** (*market, event\_id=4*)

Activate subscription on market events.

**Parameters**

- **market** (*str, bitshares.aio.Market*) – market to set subscription on
- **event\_id** (*int*) – id of this subscription in upcoming notifications

**subscribe\_to\_pending\_transactions** (*event\_id=0*)

Activate subscription to pending transactions.

Each time transaction is pushed to database an event will occur in self.notifications.

**Parameters** **event\_id** (*int*) – id of this subscription in upcoming notifications

**transfer** (*to, amount, asset, memo=”, account=None, \*\*kwargs*)

Transfer an asset to another account.

**Parameters**

- **to** (*str*) – Recipient
- **amount** (*float*) – Amount to transfer
- **asset** (*str*) – Asset to transfer
- **memo** (*str*) – (optional) Memo, may begin with # for encrypted messaging
- **account** (*str*) – (optional) the source account for the transfer if not default\_account

**tx** ()

Returns the default transaction buffer

**txbuffer**

Returns the currently active tx buffer

**unlock** (*\*args, \*\*kwargs*)

Unlock the internal wallet

**unset\_proxy** (*account=None, \*\*kwargs*)

Unset the proxy account to start voting yourself.

**update\_cer** (*symbol, cer, account=None*)

Update the Core Exchange Rate (CER) of an asset.

**Parameters**

- **symbol** (*str*) – Symbol of the asset to publish feed for
- **cer** (`bitshares.price.Price`) – Core exchange Rate
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**update\_memo\_key** (*key*, *account=None*, *\*\*kwargs*)

Update an account's memo public key.

This method does **not** add any private keys to your wallet but merely changes the memo public key.

**Parameters**

- **key** (*str*) – New memo public key
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**update\_voting\_ticket** (*ticket\_id*, *new\_target\_type*, *amount\_to\_update*, *account=None*, *\*\*kwargs*)

Update a voting ticket.

**Parameters**

- **ticket\_id** (*str*) – Id (e.g. "1.18.xxx") of the ticket to update.
- **target\_type** (*int*, *str*) – New lock period target. Should be a string from `operations.ticket_type_strings` or the index of the intended string.
- **amount\_to\_update** (`Amount`, *None*) – Amount to move over to the new lock-up target. (Optional - absence implies update whole amount.)

**update\_witness** (*witness\_identifier*, *url=None*, *key=None*, *\*\*kwargs*)

Upgrade a witness account.

**Parameters**

- **witness\_identifier** (*str*) – Identifier for the witness
- **url** (*str*) – New URL for the witness
- **key** (*str*) – Public Key for the signing

**upgrade\_account** (*account=None*, *\*\*kwargs*)

Upgrade an account to Lifetime membership.

**Parameters** **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**vesting\_balance\_withdraw** (*vesting\_id*, *amount=None*, *account=None*, *\*\*kwargs*)

Withdraw vesting balance.

**Parameters**

- **vesting\_id** (*str*) – Id of the vesting object
- **Amount** (`bitshares.amount.Amount`) – to withdraw ("all" if not provided")
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**withdraw\_from\_liquidity\_pool** (*pool*, *share\_amount*, *account=None*, *\*\*kwargs*)

Withdraw stake from a liquidity pool.

**Parameters**

- **pool** (*str*, *Asset*) – The liquidity pool to use. Can be the pool id as a string, or can be an Asset, asset\_id, or symbol of the share asset for the pool.
- **share\_amount** (*Amount*) – Amount of share asset to redeem. Must be a quantity of the pool's share\_asset.

## bitshares.aio.block module

**class** bitshares.aio.block.**Block** (\*args, \*\*kwargs)

Bases: *bitshares.aio.instance.BlockchainInstance*, *bitshares.aio.block.Block*

Read a single block from the chain.

### Parameters

- **block** (*int*) – block number
- **blockchain\_instance** (*bitshares.aio.bitshares.BitShares*) – BitShares instance
- **lazy** (*bool*) – Use lazy loading
- **loop** – async event loop

Instances of this class are dictionaries that come with additional methods (see below) that allow dealing with a block and it's corresponding functions.

```
from bitshares.aio.block import Block
block = await Block(1)
print(block)
```

### bitshares

Alias for the specific blockchain.

### blockchain

### blockchain\_instance\_class

alias of *bitshares.aio.instance.BlockchainInstance*

### classmethod cache\_object (data, key=None)

This classmethod allows to feed an object into the cache is is mostly used for testing

### chain

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

### classmethod clear\_cache ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

### define\_classes ()

Needs to define instance variables that provide classes

### fromkeys ()

Create a new dictionary with keys from iterable and values set to value.

### get ()

Return the value for key if key is in the dictionary, else default.

### get\_instance\_class ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier** = **None**

**incached** (*id*)

Is an element cached?

**classmethod inject** (*cls*)

**items** ()

This overrides items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static objectid\_valid** (*i*)

Test if a string looks like a regular object id of the form::

`xxxx.yyyyy.zzzz`

with those being numbers.

**perform\_id\_tests** = **True**

**pop** (*k*, [*d*]) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise **KeyError** is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a

2-tuple; but raise **KeyError** if D is empty.

**refresh** ()

Even though blocks never change, you freshly obtain its contents from an API with this method

**static set\_cache\_store** (*klass*, \**args*, \*\**kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters instance** (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = **1**

**store** (*data*, *key*='id')

Cache the list

**Parameters data** (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)

Alias for `objectid_valid`

**testid** (*id*)  
In contrast to validity, this method tests if the objectid matches the type\_id provided in self.type\_id or self.type\_ids

**time** ()  
Return a datetime instance for the timestamp of this block

**type\_id** = 'n/a'

**type\_ids** = []

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitshares.aio.block.**BlockHeader** (\*args, *\*\*kwargs*)  
Bases: *bitshares.aio.instance.BlockchainInstance*, *bitshares.aio.block.BlockHeader*

**bitshares**  
Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**  
alias of *bitshares.aio.instance.BlockchainInstance*

**classmethod** **cache\_object** (*data*, *key=None*)  
This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**  
Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod** **clear\_cache** ()  
Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()  
Needs to define instance variables that provide classes

**fromkeys** ()  
Create a new dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()  
Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)  
Get an element from the cache explicitly

**identifier** = None

**incached** (*id*)  
Is an element cached?

**classmethod** **inject** (*cls*)

**items()**

This overrides items() so that refresh() is called if the object is not already fetched

**keys()** → a set-like object providing a view on D's keys

**static objectid\_valid(i)**

Test if a string looks like a regular object id of the form::

`xxxx.yyyyy.zzzz`

with those being numbers.

**perform\_id\_tests = True**

**pop(k[, d])** → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

**popitem()** → (k, v), remove and return some (key, value) pair as a 2-tuple; but raise KeyError if D is empty.

**refresh()**

Even though blocks never change, you freshly obtain its contents from an API with this method

**static set\_cache\_store(klass, \*args, \*\*kwargs)**

**classmethod set\_shared\_blockchain\_instance(instance)**

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters instance(chaininstance)** – Chain instance

**classmethod set\_shared\_config(config)**

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id = 1**

**store(data, key='id')**

Cache the list

**Parameters data(list)** – List of objects to cache

**test\_valid\_objectid(i)**

Alias for `objectid_valid`

**testid(id)**

In contrast to validity, this method tests if the objectid matches the type\_id provided in `self.type_id` or `self.type_ids`

**time()**

Return a datetime instance for the timestamp of this block

**type\_id = 'n/a'**



**type\_ids** = []

**update** ([*E*], **\*\*F**) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

## bitshares.aio.blockchain module

**class** bitshares.aio.blockchain.**Blockchain** (\*args, \*\*kwargs)

Bases: *bitshares.aio.instance.BlockchainInstance*, *bitshares.aio.blockchain.Blockchain*

This class allows to access the blockchain and read data from it.

### Parameters

- **blockchain\_instance** (*bitshares.aio.bitshares.BitShares*) – BitShares instance
- **mode** (*str*) – (default) Irreversible block (*irreversible*) or actual head block (*head*)
- **max\_block\_wait\_repetition** (*int*) – (default) 3 maximum wait time for next block  
is `max_block_wait_repetition * block_interval`

This class let's you deal with blockchain related data and methods.

**awaitTxConfirmation** (*transaction*, *limit=10*)

Returns the transaction as seen by the blockchain after being included into a block

---

**Note:** If you want instant confirmation, you need to instantiate class: *blockchain.Blockchain* with `mode="head"`, otherwise, the call will wait until confirmed in an irreversible block.

---



---

**Note:** This method returns once the blockchain has included a transaction with the **same signature**. Even though the signature is not usually used to identify a transaction, it still cannot be forfeited and is derived from the transaction content and thus identifies a transaction uniquely.

---

## bitshares

Alias for the specific blockchain.

**block\_time** (*block\_num*)

Returns a datetime of the block with the given block number.

**Parameters** **block\_num** (*int*) – Block number

**block\_timestamp** (*block\_num*)

Returns the timestamp of the block with the given block number.

**Parameters** **block\_num** (*int*) – Block number

## blockchain

**blockchain\_instance\_class**

alias of *bitshares.aio.instance.BlockchainInstance*

**blocks** (*start=None*, *stop=None*)

Yields blocks starting from *start*.

**Parameters**

- **start** (*int*) – Starting block
- **stop** (*int*) – Stop at this block
- **mode** (*str*) – We here have the choice between “head” (the last block) and “irreversible” (the block that is confirmed by 2/3 of all block producers and is thus irreversible)

**chain**

Short form for blockchain (for the lazy)

**chainParameters** ()

The blockchain parameters, such as fees, and committee-controlled parameters are returned here

**config** ()

Returns object 2.0.0

**define\_classes** ()

Needs to define instance variables that provide classes

**get\_all\_accounts** (*start=*”, *stop=*”, *steps=1000.0*, *\*\*kwargs*)

Yields account names between start and stop.

**Parameters**

- **start** (*str*) – Start at this account name
- **stop** (*str*) – Stop at this account name
- **steps** (*int*) – Obtain *steps* ret with a single call from RPC

**get\_block\_interval** ()

This call returns the block interval

**get\_chain\_properties** ()

Return chain properties

**get\_current\_block** ()

This call returns the current block

---

**Note:** The block number returned depends on the *mode* used when instantiating from this class.

---

**get\_current\_block\_num** ()

This call returns the current block

---

**Note:** The block number returned depends on the *mode* used when instantiating from this class.

---

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**get\_network** ()

Identify the network

**Returns** Network parameters

**Return type** dict

**info** ()

This call returns the *dynamic global properties*

**classmethod inject** (*cls*)

**is\_irreversible\_mode()**

**ops** (*start=None, stop=None, \*\*kwargs*)

Yields all operations (excluding virtual operations) starting from *start*.

#### Parameters

- **start** (*int*) – Starting block
- **stop** (*int*) – Stop at this block
- **mode** (*str*) – We here have the choice between “head” (the last block) and “irreversible” (the block that is confirmed by 2/3 of all block producers and is thus irreversible)
- **only\_virtual\_ops** (*bool*) – Only yield virtual operations

This call returns a list that only carries one operation and its type!

**participation\_rate**

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** **instance** (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**stream** (*opNames=[], \*args, \*\*kwargs*)

Yield specific operations (e.g. comments) only

#### Parameters

- **opNames** (*array*) – List of operations to filter for
- **start** (*int*) – Start at this block
- **stop** (*int*) – Stop at this block
- **mode** (*str*) – We here have the choice between \* “head”: the last block \* “irreversible”: the block that is confirmed by 2/3 of all

block producers and is thus irreversible!

The dict output is formatted such that `type` carries the operation type, `timestamp` and `block_num` are taken from the block the operation was stored in and the other key depend on the actualy operation.

**update\_chain\_parameters()**

**wait\_for\_and\_get\_block** (*block\_number, blocks\_waiting\_for=None*)

Get the desired block from the chain, if the current head block is smaller (for both head and irreversible) then we wait, but a maximum of `blocks_waiting_for * max_block_wait_repetition` time before failure.

#### Parameters

- **block\_number** (*int*) – desired block number
- **blocks\_waiting\_for** (*int*) – (default) difference between `block_number` and current head how many blocks we are willing to wait, positive int

**bitshares.aio.blockchainobject module**

```
class bitshares.aio.blockchainobject.BlockchainObject (*args, **kwargs)
    Bases: bitshares.aio.instance.BlockchainInstance, bitshares.aio.
    blockchainobject.BlockchainObject

    bitshares
        Alias for the specific blockchain.

    blockchain

    blockchain_instance_class
        alias of bitshares.aio.instance.BlockchainInstance

    classmethod cache_object (data, key=None)
        This classmethod allows to feed an object into the cache is is mostly used for testing

    chain
        Short form for blockchain (for the lazy)

    clear () → None. Remove all items from D.

    classmethod clear_cache ()
        Clear/Reset the entire Cache

    copy () → a shallow copy of D

    define_classes ()
        Needs to define instance variables that provide classes

    fromkeys ()
        Create a new dictionary with keys from iterable and values set to value.

    get ()
        Return the value for key if key is in the dictionary, else default.

    get_instance_class ()
        Should return the Chain instance class, e.g. bitshares.BitShares

    getfromcache (id)
        Get an element from the cache explicitly

    identifier = None

    incached (id)
        Is an element cached?

    classmethod inject (cls)

    items ()
        This overrides items() so that refresh() is called if the object is not already fetched

    keys () → a set-like object providing a view on D's keys

    static objectid_valid (i)
        Test if a string looks like a regular object id of the form::



xxxx.yyzz.zzzz



        with those being numbers.

    perform_id_tests = True
```

**pop** ( $k, d$ )  $\rightarrow v$ , remove specified key and return the corresponding value.  
If key is not found, d is returned if given, otherwise `KeyError` is raised

**popitem** ()  $\rightarrow (k, v)$ , remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if D is empty.

**static set\_cache\_store** (*klass*, \*args, \*\*kwargs)

**classmethod set\_shared\_blockchain\_instance** (*instance*)  
This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters instance** (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)  
This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()  
This method allows to set the current instance as default

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()  
This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (*data*, *key*='id')  
Cache the list

**Parameters data** (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)  
Alias for `objectid_valid`

**testid** (*id*)  
In contrast to validity, this method tests if the objectid matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id** = None

**type\_ids** = []

**update** ( $[E]$ , \*\*F)  $\rightarrow$  None. Update D from dict/iterable E and F.  
If E is present and has a `.keys()` method, then does: for k in E: `D[k] = E[k]` If E is present and lacks a `.keys()` method, then does: for k, v in E: `D[k] = v` In either case, this is followed by: for k in F: `D[k] = F[k]`

**values** ()  $\rightarrow$  an object providing a view on D's values

**class** `bitshares.aio.blockchainobject.Object` (\*args, \*\*kwargs)  
Bases: `bitshares.aio.instance.BlockchainInstance`, `bitshares.aio.blockchainobject.Object`

**bitshares**  
Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**  
alias of `bitshares.aio.instance.BlockchainInstance`

**classmethod** `cache_object (data, key=None)`

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod** `clear_cache ()`

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier** = None

**incached** (*id*)

Is an element cached?

**classmethod** `inject (cls)`

**items** ()

This overrides items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static** `objectid_valid (i)`

Test if a string looks like a regular object id of the form::

`xxxx.yyzz.zzzz`

with those being numbers.

**perform\_id\_tests** = False

**pop** (*k*, [*d*]) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a

2-tuple; but raise `KeyError` if D is empty.

**refresh** ()

This is the refresh method that overloads the prototype in `BlockchainObject`.

**static** `set_cache_store (klass, *args, **kwargs)`

**classmethod** `set_shared_blockchain_instance (instance)`

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** `instance (chaininstance)` – Chain instance

**classmethod set\_shared\_config** (*config*)  
 This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()  
 This method allows to set the current instance as default

**setdefault** ()  
 Insert key with a value of default if key is not in the dictionary.  
 Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()  
 This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (*data*, *key*='id')  
 Cache the list

**Parameters** *data* (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)  
 Alias for `objectid_valid`

**testid** (*id*)  
 In contrast to validity, this method tests if the `objectid` matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id** = None

**type\_ids** = []

**update** ([*E*], \*\**F*) → None. Update D from dict/iterable E and F.  
 If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

## bitshares.aio.committee module

**class** `bitshares.aio.committee.Committee` (*\*args*, \*\**kwargs*)  
 Bases: `bitshares.aio.instance.BlockchainInstance`, `bitshares.aio.committee.Committee`

Read data about a Committee Member in the chain.

### Parameters

- **member** (*str*) – Name of the Committee Member
- **blockchain\_instance** (*bitshares*) – BitShares() instance to use when accessing a RPC
- **lazy** (*bool*) – Use lazy loading

**account**

**account\_id**

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of *bitshares.aio.instance.BlockchainInstance*

**classmethod cache\_object** (*data, key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier = None**

**incached** (*id*)

Is an element cached?

**classmethod inject** (*cls*)

**items** ()

This overrides items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static objectid\_valid** (*i*)

Test if a string looks like a regular object id of the form::

`xxxx.yyzzz.zzzz`

with those being numbers.

**perform\_id\_tests = True**

**pop** (*k[, d]*) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise *KeyError* is raised

**popitem** () → (*k, v*), remove and return some (key, value) pair as a

2-tuple; but raise *KeyError* if *D* is empty.

**refresh** ()

**static set\_cache\_store** (*klass, \*args, \*\*kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of *SharedInstance.instance*.



**Parameters** `instance` (*chaininstance*) – Chain instance

**classmethod** `set_shared_config` (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (*data*, *key*='id')

Cache the list

**Parameters** `data` (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)

Alias for `objectid_valid`

**testid** (*id*)

In contrast to validity, this method tests if the objectid matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id** = None

**type\_ids** = []

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

## bitshares.aio.dex module

**class** `bitshares.aio.dex.Dex` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.aio.instance.BlockchainInstance`

This class simplifies interactions with the decentralized exchange.

**Parameters** `blockchain_instance` (`bitshares.aio.bitshares.BitShares`) – Bit-Shares instance

---

**Note:** The methods of this class only deal with a single asset (at most). If you are looking to deal with orders for trading, please use `bitshares.aio.market.Market`.

---

**adjust\_collateral\_ratio** (*symbol*, *new\_collateral\_ratio*, *account=None*, *target\_collateral\_ratio=None*)

Adjust the collateral ratio of a debt position.

**Parameters**

- **symbol** (*str*) – Symbol to adjust collateral for
- **new\_collateral\_ratio** (*float*) – desired collateral ratio
- **target\_collateral\_ratio** (*float*) – Tag the call order so that in case of margin call, only enough debt is covered to get back to this ratio

**Raises**

- **ValueError** – if symbol is not a bitasset
- **ValueError** – if collateral ratio is smaller than maintenance collateral ratio
- **ValueError** – if required amounts of collateral are not available

**adjust\_debt** (*delta*, *new\_collateral\_ratio=None*, *account=None*, *target\_collateral\_ratio=None*)

Adjust the amount of debt for an asset.

**Parameters**

- **delta** (*Amount*) – Delta amount of the debt (-10 means reduce debt by 10, +10 means borrow another 10)
- **new\_collateral\_ratio** (*float*) – collateral ratio to maintain (optional, by default tries to maintain old ratio)
- **target\_collateral\_ratio** (*float*) – Tag the call order so that in case of margin call, only enough debt is covered to get back to this ratio

**Raises**

- **ValueError** – if symbol is not a bitasset
- **ValueError** – if collateral ratio is smaller than maintenance collateral ratio
- **ValueError** – if required amounts of collateral are not available

**bitshares**

Alias for the specific blockchain.

**blockchain**

**borrow** (*amount*, *collateral\_ratio=None*, *account=None*, *target\_collateral\_ratio=None*)

Borrow bitassets/smartcoins from the network by putting up collateral in a CFD at a given collateral ratio.

**Parameters**

- **amount** (*Amount*) – Amount to borrow (denoted in ‘asset’)
- **collateral\_ratio** (*float*) – Collateral ratio to borrow at
- **target\_collateral\_ratio** (*float*) – Tag the call order so that in case of margin call, only enough debt is covered to get back to this ratio

**Raises**

- **ValueError** – if symbol is not a bitasset
- **ValueError** – if collateral ratio is smaller than maintenance collateral ratio
- **ValueError** – if required amounts of collateral are not available

**chain**

Short form for blockchain (for the lazy)

**close\_debt\_position** (*symbol*, *account=None*)

Close a debt position and reclaim the collateral.

**Parameters** `symbol` (*str*) – Symbol to close debt position for

**Raises** `ValueError` – if symbol has no open call position

**define\_classes** ()

Needs to define instance variables that provide classes

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject** (*cls*)

**list\_debt\_positions** (*account=None*)

List Call Positions (borrowed assets and amounts)

**Returns** Struct of assets with amounts and call price

**Return type** dict

**Example:**

**returnFees** ()

Returns a dictionary of all fees that apply through the network.

Example output:

```
{'proposal_create': {'fee': 400000.0},
'asset_publish_feed': {'fee': 1000.0}, 'account_create':
{'basic_fee': 950000.0, 'price_per_kbyte': 20000.0,
'premium_fee': 40000000.0}, 'custom': {'fee': 20000.0},
'asset_fund_fee_pool': {'fee': 20000.0},
'override_transfer': {'fee': 400000.0}, 'fill_order':
{}, 'asset_update': {'price_per_kbyte': 20000.0, 'fee':
200000.0}, 'asset_update_feed_producers': {'fee':
10000000.0}, 'assert': {'fee': 20000.0},
'committee_member_create': {'fee': 100000000.0}}
```

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** `instance` (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

## bitshares.ait.genesisbalance module

**class** `bitshares.ait.genesisbalance.GenesisBalance` (*\*args, \*\*kwargs*)

Bases: `bitshares.ait.instance.BlockchainInstance`, `bitshares.ait.genesisbalance.GenesisBalance`

Read data about a Genesis Balances from the chain.

**Parameters**

- **identifier** (*str*) – identifier of the balance
- **blockchain\_instance** (*bitshares*) – bitshares() instance to use when accessing a RPC

**bitshares**

Alias for the specific blockchain.

**blockchain****blockchain\_instance\_class**

alias of *bitshares.aio.instance.BlockchainInstance*

**classmethod cache\_object** (*data, key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**claim** (*account=None, \*\*kwargs*)

Claim a balance from the genesis block

**Parameters**

- **balance\_id** (*str*) – The identifier that identifies the balance to claim (1.15.x)
- **account** (*str*) – (optional) the account that owns the bet (defaults to *default\_account*)

**clear** () → None. Remove all items from D.

**classmethod clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier** = None

**incached** (*id*)

Is an element cached?

**classmethod inject** (*cls*)**items** ()

This overrides items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static objectid\_valid** (*i*)

Test if a string looks like a regular object id of the form::

```
xxxx.yyyyy.zzzz
```

with those being numbers.

**perform\_id\_tests** = True

**pop** (*k*, *d*) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if *D* is empty.

**refresh** ()

**static set\_cache\_store** (*klass*, *\*args*, *\*\*kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (*data*, *key*=*'id'*)

Cache the list

**Parameters** *data* (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)

Alias for `objectid_valid`

**testid** (*id*)

In contrast to validity, this method tests if the objectid matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id** = 15

**type\_ids** = []

**update** (*[E]*, *\*\*F*) → None. Update *D* from dict/iterable *E* and *F*.

If *E* is present and has a `.keys()` method, then does: for *k* in *E*: *D*[*k*] = *E*[*k*] If *E* is present and lacks a `.keys()` method, then does: for *k*, *v* in *E*: *D*[*k*] = *v* In either case, this is followed by: for *k* in *F*: *D*[*k*] = *F*[*k*]

**values** () → an object providing a view on *D*'s values

**class** `bitshares.aio.genesisbalance.GenesisBalances` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.aio.instance.BlockchainInstance`, `bitshares.aio.genesisbalance.GenesisBalances`

List genesis balances that can be claimed from the keys in the wallet.

**append()**

Append object to the end of the list.

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of *bitshares.aio.instance.BlockchainInstance*

**chain**

Short form for blockchain (for the lazy)

**clear()**

Remove all items from list.

**copy()**

Return a shallow copy of the list.

**count()**

Return number of occurrences of value.

**define\_classes()**

Needs to define instance variables that provide classes

**extend()**

Extend list by appending elements from the iterable.

**get\_instance\_class()**

Should return the Chain instance class, e.g. *bitshares.BitShares*

**index()**

Return first index of value.

Raises ValueError if the value is not present.

**classmethod inject(cls)**

**insert()**

Insert object before index.

**pop()**

Remove and return item at index (default last).

Raises IndexError if list is empty or index is out of range.

**remove()**

Remove first occurrence of value.

Raises ValueError if the value is not present.

**reverse()**

Reverse *IN PLACE*.

**classmethod set\_shared\_blockchain\_instance(instance)**

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters** *instance(chaininstance)* – Chain instance

**classmethod set\_shared\_config(config)**

This allows to set a config that will be used when calling *shared\_blockchain\_instance* and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**sort()**

Stable sort *IN PLACE*.

## bitshares.aio.htlc module

**class** `bitshares.aio.htlc.Htlc(*args, **kwargs)`

Bases: `bitshares.aio.blockchainobject.BlockchainObject`

Read data about an HTLC contract on the chain.

### Parameters

- **id**(*str*) – id of the HTLC
- **blockchain\_instance**(*bitshares*) – BitShares() instance to use when accessing a RPC

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.aio.instance.BlockchainInstance`

**classmethod** `cache_object(data, key=None)`

This classmethod allows to feed an object into the cache is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear()** → None. Remove all items from D.

**classmethod** `clear_cache()`

Clear/Reset the entire Cache

**copy()** → a shallow copy of D

**define\_classes()**

Needs to define instance variables that provide classes

**fromkeys()**

Create a new dictionary with keys from iterable and values set to value.

**get()**

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class()**

Should return the Chain instance class, e.g. `bitshares.BitShares`

**getfromcache**(*id*)

Get an element from the cache explicitly

**identifier = None**

**incached** (*id*)

Is an element cached?

**classmethod inject** (*cls*)

**items** ()

This overrides items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static objectid\_valid** (*i*)

Test if a string looks like a regular object id of the form::

`xxxx.yyzzz.zzzz`

with those being numbers.

**perform\_id\_tests** = True

**pop** (*k*, [*d*]) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if D is empty.

**refresh** ()

**static set\_cache\_store** (*klass*, \**args*, \*\**kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters instance** (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (*data*, *key*='id')

Cache the list

**Parameters data** (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)

Alias for `objectid_valid`

**testid** (*id*)

In contrast to validity, this method tests if the objectid matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id** = 16



**type\_ids** = []

**update** ([*E*], **\*\*F**) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

## bitshares.aio.instance module

**class** bitshares.aio.instance.**BlockchainInstance** (\*args, **\*\*kwargs**)

Bases: graphenecommon.aio.instance.AbstractBlockchainInstanceProvider

This is a class that allows compatibility with previous naming conventions.

**bitshares**

Alias for the specific blockchain.

**blockchain**

**chain**

Short form for blockchain (for the lazy)

**define\_classes** ()

Needs to define instance variables that provide classes

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject** (cls)

**classmethod set\_shared\_blockchain\_instance** (instance)

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters** *instance* (chaininstance) – Chain instance

**classmethod set\_shared\_config** (config)

This allows to set a config that will be used when calling *shared\_blockchain\_instance* and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**shared\_blockchain\_instance** ()

This method will initialize *SharedInstance.instance* and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**class** bitshares.aio.instance.**SharedInstance**

Bases: object

This class merely offers a singleton for the Blockchain Instance.

**config** = {}

**instance** = None

bitshares.aio.instance.**set\_shared\_bitshares\_instance** (instance)

bitshares.aio.instance.**set\_shared\_blockchain\_instance** (instance)

bitshares.aio.instance.**set\_shared\_config** (config)

bitshares.aio.instance.**shared\_bitshares\_instance** ()

```
bitshares.aio.instance.shared_blockchain_instance()
```

## bitshares.aio.market module

```
class bitshares.aio.market.Market(*args, **kwargs)
```

Bases: *bitshares.aio.instance.BlockchainInstance*, *bitshares.aio.market.Market*

This class allows to easily access Markets on the blockchain for trading, etc.

### Parameters

- **blockchain\_instance** (*bitshares.aio.bitshares.BitShares*) – BitShares instance
- **base** (*bitshares.aio.asset.Asset*) – Base asset
- **quote** (*bitshares.aio.asset.Asset*) – Quote asset

**Returns** Blockchain Market

**Return type** dictionary with overloaded methods

Instances of this class are dictionaries that come with additional methods (see below) that allow dealing with a market and it's corresponding functions.

This class tries to identify **two** assets as provided in the parameters in one of the following forms:

- base and quote are valid assets (according to *bitshares.asset.Asset*)
- base:quote separated with :
- base/quote separated with /
- base-quote separated with –

---

**Note:** Throughout this library, the quote symbol will be presented first (e.g. USD:BTS with USD being the quote), while the base only refers to a secondary asset for a trade. This means, if you call *bitshares.market.Market.sell()* or *bitshares.market.Market.buy()*, you will sell/buy **only quote** and obtain/pay **only base**.

---

```
accountopenorders (account=None)
```

Returns open Orders.

**Parameters** **account** (*bitshares.account.Account*) – Account name or instance of Account to show orders for in this market

```
accounttrades (account=None, limit=25)
```

Returns your trade history for a given market, specified by the “currencyPair” parameter. You may also specify “all” to get the orderbooks of all markets.

### Parameters

- **currencyPair** (*str*) – Return results for a particular market only (default: “all”)
- **limit** (*int*) – Limit the amount of orders (default: 25)

Output Parameters:

- *type*: sell or buy
- *rate*: price for *quote* denoted in *base* per *quote*
- *amount*: amount of quote

- *total*: amount of base at asked price (amount/price)

---

**Note:** This call goes through the trade history and searches for your account, if there are no orders within `limit` trades, this call will return an empty array.

---

### **bitshares**

Alias for the specific blockchain.

### **blockchain**

### **blockchain\_instance\_class**

alias of `bitshares.aio.instance.BlockchainInstance`

**buy** (*price*, *amount*, *expiration=None*, *killfill=False*, *account=None*, *returnOrderId=False*, *\*\*kwargs*)

Places a buy order in a given market.

#### **Parameters**

- **price** (*float*) – price denoted in base/quote
- **amount** (*number*) – Amount of quote to buy
- **expiration** (*number*) – (optional) expiration time of the order in seconds (defaults to 7 days)
- **killfill** (*bool*) – flag that indicates if the order shall be killed if it is not filled (defaults to False)
- **account** (*string*) – Account name that executes that order
- **returnOrderId** (*string*) – If set to “head” or “irreversible” the call will wait for the tx to appear in the head/irreversible block and add the key “orderid” to the tx output

Prices/Rates are denoted in ‘base’, i.e. the USD\_BTS market is priced in BTS per USD.

**Example:** in the USD\_BTS market, a price of 300 means a USD is worth 300 BTS

---

**Note:** All prices returned are in the **reversed** orientation as the market. I.e. in the BTC/BTS market, prices are BTS per BTC. That way you can multiply prices with *1.05* to get a +5%.

---

**Warning:** Since buy orders are placed as limit-sell orders for the base asset, you may end up obtaining more of the buy asset than you placed the order for. Example:

- You place and order to buy 10 USD for 100 BTS/USD
- This means that you actually place a sell order for 1000 BTS in order to obtain **at least** 10 USD
- If an order on the market exists that sells USD for cheaper, you will end up with more than 10 USD

**cancel** (*orderNumber*, *account=None*, *\*\*kwargs*)

Cancels an order you have placed in a given market. Requires only the “orderNumber”. An order number takes the form `1.7.xxx`.

**Parameters** **orderNumber** (*str*) – The Order Object ide of the form `1.7.xxxx`

### **chain**

Short form for blockchain (for the lazy)

**clear()** → None. Remove all items from D.

**copy()** → a shallow copy of D

**core\_base\_market()**

This returns an instance of the market that has the core market of the base asset.

It means that base needs to be a market pegged asset and returns a market to it's collateral asset.

**core\_quote\_market()**

This returns an instance of the market that has the core market of the quote asset.

It means that quote needs to be a market pegged asset and returns a market to it's collateral asset.

**define\_classes()**

Needs to define instance variables that provide classes

**fromkeys()**

Create a new dictionary with keys from iterable and values set to value.

**get()**

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class()**

Should return the Chain instance class, e.g. *bitshares.BitShares*

**get\_limit\_orders(limit=25)**

Returns the list of limit orders for a given market.

**Parameters** **limit** (*int*) – Limit the amount of orders (default: 25)

Sample output:

```
[0.003679 USD/BTS (1.9103 USD|519.29602 BTS),
0.003676 USD/BTS (299.9997 USD|81606.16394 BTS),
0.003665 USD/BTS (288.4618 USD|78706.21881 BTS),
0.003665 USD/BTS (3.5285 USD|962.74409 BTS),
0.003665 USD/BTS (72.5474 USD|19794.41299 BTS)] ,
```

---

**Note:** Each bid is an instance of class:*bitshares.price.Order* and thus carries the keys *base*, *quote* and *price*. From those you can obtain the actual amounts for sale

---

**get\_string(separator=':')**

Return a formatted string that identifies the market, e.g. USD:BTS

**Parameters** **separator** (*str*) – The separator of the assets (defaults to :)

**classmethod inject(cls)**

**items()** → a set-like object providing a view on D's items

**keys()** → a set-like object providing a view on D's keys

**orderbook(limit=25)**

Returns the order book for a given market. You may also specify "all" to get the orderbooks of all markets.

**Parameters** **limit** (*int*) – Limit the amount of orders (default: 25)

Sample output:

```
{'bids': [0.003679 USD/BTS (1.9103 USD|519.29602 BTS),
0.003676 USD/BTS (299.9997 USD|81606.16394 BTS),
0.003665 USD/BTS (288.4618 USD|78706.21881 BTS),
0.003665 USD/BTS (3.5285 USD|962.74409 BTS),
0.003665 USD/BTS (72.5474 USD|19794.41299 BTS)],
'asks': [0.003738 USD/BTS (36.4715 USD|9756.17339 BTS),
0.003738 USD/BTS (18.6915 USD|5000.00000 BTS),
0.003742 USD/BTS (182.6881 USD|48820.22081 BTS),
0.003772 USD/BTS (4.5200 USD|1198.14798 BTS),
0.003799 USD/BTS (148.4975 USD|39086.59741 BTS)]}
```

---

**Note:** Each bid is an instance of class:*bitshares.price.Order* and thus carries the keys `base`, `quote` and `price`. From those you can obtain the actual amounts for sale

---



---

**Note:** This method does order consolidation and hides some details of individual orders!

---

**pop** (*k*, [*d*]) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if *D* is empty.

**sell** (*price*, *amount*, *expiration=None*, *killfill=False*, *account=None*, *returnOrderId=False*, *\*\*kwargs*)

Places a sell order in a given market.

#### Parameters

- **price** (*float*) – price denoted in `base/quote`
- **amount** (*number*) – Amount of `quote` to sell
- **expiration** (*number*) – (optional) expiration time of the order in seconds (defaults to 7 days)
- **killfill** (*bool*) – flag that indicates if the order shall be killed if it is not filled (defaults to `False`)
- **account** (*string*) – Account name that executes that order
- **returnOrderId** (*string*) – If set to “head” or “irreversible” the call will wait for the tx to appear in the head/irreversible block and add the key “orderid” to the tx output

Prices/Rates are denoted in ‘base’, i.e. the USD\_BTS market is priced in BTS per USD.

**Example:** in the USD\_BTS market, a price of 300 means a USD is worth 300 BTS

---

**Note:** All prices returned are in the **reversed** orientation as the market. I.e. in the BTC/BTS market, prices are BTS per BTC. That way you can multiply prices with *1.05* to get a +5%.

---

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**ticker()**

Returns the ticker for all markets.

Output Parameters:

- **last**: Price of the order last filled
- **lowestAsk**: Price of the lowest ask
- **highestBid**: Price of the highest bid
- **baseVolume**: Volume of the base asset
- **quoteVolume**: Volume of the quote asset
- **percentChange**: 24h change percentage (in %)
- **settlement\_price**: Settlement Price for borrow/settlement
- **core\_exchange\_rate**: Core exchange rate for payment of fee in non-BTS asset
- **price24h**: the price 24h ago

Sample Output:

```
{
  {
    "quoteVolume": 48328.73333,
    "quoteSettlement_price": 332.3344827586207,
    "lowestAsk": 340.0,
    "baseVolume": 144.1862,
    "percentChange": -1.9607843231354893,
    "highestBid": 334.20000000000005,
    "latest": 333.3333330133934,
  }
}
```

**trades** (*limit=25, start=None, stop=None*)

Returns your trade history for a given market.

#### Parameters

- **limit** (*int*) – Limit the amount of orders (default: 25)
- **start** (*datetime*) – start time
- **stop** (*datetime*) – stop time

**update** (*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**volume24h()**

Returns the 24-hour volume for all markets, plus totals for primary currencies.

Sample output:

```
{
  "BTS": 361666.63617,
  "USD": 1087.0
}
```

**bitshares.aio.memo module**

**class** bitshares.aio.memo.Memo(\*args, \*\*kwargs)

Bases: *bitshares.aio.instance.BlockchainInstance*, *bitshares.aio.memo.Memo*

Deals with Memos that are attached to a transfer.

**Parameters**

- **from\_account** (*bitshares.aio.account.Account*) – Account that has sent the memo
- **to\_account** (*bitshares.aio.account.Account*) – Account that has received the memo
- **blockchain\_instance** (*bitshares.aio.bitshares.BitShares*) – BitShares instance

A memo is encrypted with a shared secret derived from a private key of the sender and a public key of the receiver. Due to the underlying mathematics, the same shared secret can be derived by the private key of the receiver and the public key of the sender. The encrypted message is perturbed by a nonce that is part of the transmitted message.

```
from bitshares.aio.memo import Memo
m = await Memo("bitshareseu", "wallet.xeroc")
m.unlock_wallet("secret")
enc = (m.encrypt("foobar"))
print(enc)
>> {'nonce': '17329630356955254641', 'message': '8563e2bb2976e0217806d642901a2855
↪ '}
print(m.decrypt(enc))
>> foobar
```

To decrypt a memo, simply use

```
from bitshares.aio.memo import Memo
m = await Memo()
m.blockchain.wallet.unlock("secret")
print(memo.decrypt(op_data["memo"]))
```

if `op_data` being the payload of a transfer operation.

**bitshares**

Alias for the specific blockchain.

**blockchain****blockchain\_instance\_class**

alias of *bitshares.aio.instance.BlockchainInstance*

**chain**

Short form for blockchain (for the lazy)

**decrypt** (*message*)

Decrypt a message

**Parameters** **message** (*dict*) – encrypted memo message

**Returns** decrypted message

**Return type** str

**define\_classes** ()

Needs to define instance variables that provide classes

**encrypt** (*message*)

Encrypt a memo

**Parameters** **message** (*str*) – clear text memo message

**Returns** encrypted message

**Return type** str

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject** (*cls*)**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** **instance** (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**unlock\_wallet** (*\*args, \*\*kwargs*)

Unlock the library internal wallet

## bitshares.aio.message module

**class** bitshares.aio.message.**Message** (*\*args, \*\*kwargs*)

Bases: `bitshares.aio.instance.BlockchainInstance`, `bitshares.aio.message.Message`

**MESSAGE\_SPLIT** = ('-----BEGIN BITSHARES SIGNED MESSAGE-----', '-----BEGIN META-----', '-----END META-----', '-----END BITSHARES SIGNED MESSAGE-----')

**SIGNED\_MESSAGE\_ENCAPSULATED** = '\n{MESSAGE\_SPLIT[0]}\n{message}\n{MESSAGE\_SPLIT[1]}\n{naccount}\n{MESSAGE\_SPLIT[2]}\n{message}\n{MESSAGE\_SPLIT[3]}\n{nmemokey}\n{MESSAGE\_SPLIT[4]}\n{nmemokey}'

**SIGNED\_MESSAGE\_META** = '{message}\naccount={meta[account]}\nmemokey={meta[memokey]}\nbl'

**bitshares**

Alias for the specific blockchain.

**blockchain**



**blockchain\_instance\_class**alias of `bitshares.aio.instance.BlockchainInstance`**chain**

Short form for blockchain (for the lazy)

**define\_classes()**

Needs to define instance variables that provide classes

**get\_instance\_class()**Should return the Chain instance class, e.g. `bitshares.BitShares`**classmethod inject(cls)****classmethod set\_shared\_blockchain\_instance(instance)**This method allows us to override default instance for all users of `SharedInstance.instance`.**Parameters** `instance` (`chaininstance`) – Chain instance**classmethod set\_shared\_config(config)**This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance**set\_shared\_instance()**

This method allows to set the current instance as default

**shared\_blockchain\_instance()**This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.**sign(\*args, \*\*kwargs)**

Sign a message with an account's memo key

**Parameters** `account` (`str`) – (optional) the account that owns the bet (defaults to `default_account`)**Raises** `ValueError` – If not account for signing is provided**Returns** the signed message encapsulated in a known format**supported\_formats** = (`<class 'graphenecommon.aio.message.MessageV1'>`, `<class 'graphenecommon.aio.message.MessageV2'>`)**valid\_exceptions** = (`<class 'graphenecommon.exceptions.AccountDoesNotExistException'>`, `<class 'graphenecommon.exceptions.InvalidMessageSignature'>`)**verify(\*\*kwargs)**

Verify a message with an account's memo key

**Parameters** `account` (`str`) – (optional) the account that owns the bet (defaults to `default_account`)**Returns** True if the message is verified successfully:raises `InvalidMessageSignature` if the signature is not ok**bitshares.aio.price module****class** `bitshares.aio.price.FilledOrder` (`order`, `**kwargs`)Bases: `bitshares.aio.price.Price`This class inherits `bitshares.aio.price.Price` but has the base and quote Amounts not only be used to represent the price (as a ratio of base and quote) but instead has those amounts represent the amounts of an actually filled order!

**Parameters** `blockchain_instance` (`bitshares.aio.bitshares.BitShares`) – BitShares instance

---

**Note:** Instances of this class come with an additional `time` key that shows when the order has been filled!

---

**as\_base** (*base*)

Returns the price instance so that the base asset is *base*.

Note: This makes a copy of the object!

**as\_quote** (*quote*)

Returns the price instance so that the quote asset is *quote*.

Note: This makes a copy of the object!

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.aio.instance.BlockchainInstance`

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. `bitshares.BitShares`

**classmethod inject** (*cls*)

**invert** ()

Invert the price (e.g. go from USD/BTS into BTS/USD)

**items** () → a set-like object providing a view on D's items

**json** ()

```
return { "base": self["base"].json(), "quote": self["quote"].json()
}
```

**keys** () → a set-like object providing a view on D's keys

**market**

Open the corresponding market.

**Returns** Instance of `bitshares.aio.market.Market` for the corresponding pair of assets.

**pop** (*k*, *d*) → *v*, remove specified key and return the corresponding value.  
If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a  
2-tuple; but raise `KeyError` if *D* is empty.

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**symbols** ()

**update** (*E*, *\*\*F*) → `None`. Update *D* from dict/iterable *E* and *F*.

If *E* is present and has a `.keys()` method, then does: for *k* in *E*: *D*[*k*] = *E*[*k*] If *E* is present and lacks a `.keys()` method, then does: for *k*, *v* in *E*: *D*[*k*] = *v* In either case, this is followed by: for *k* in *F*: *D*[*k*] = *F*[*k*]

**values** () → an object providing a view on *D*'s values

**class** `bitshares.aio.price.Order` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.aio.price.Price`

This class inherits `bitshares.aio.price.Price` but has the base and quote Amounts not only be used to represent the price (as a ratio of base and quote) but instead has those amounts represent the amounts of an actual order!

**Parameters** *blockchain\_instance* (`bitshares.aio.bitshares.BitShares`) – Bit-Shares instance

---

**Note:** If an order is marked as deleted, it will carry the 'deleted' key which is set to `True` and all other data be `None`.

---

**as\_base** (*base*)

Returns the price instance so that the base asset is *base*.

Note: This makes a copy of the object!

**as\_quote** (*quote*)

Returns the price instance so that the quote asset is *quote*.

Note: This makes a copy of the object!

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of *bitshares.aio.instance.BlockchainInstance*

**chain**

Short form for blockchain (for the lazy)

**clear**() → None. Remove all items from D.

**copy**() → a shallow copy of D

**define\_classes**()

Needs to define instance variables that provide classes

**fromkeys**()

Create a new dictionary with keys from iterable and values set to value.

**get**()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class**()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject**(cls)**invert**()

Invert the price (e.g. go from USD/BTS into BTS/USD)

**items**() → a set-like object providing a view on D's items

**json**()

```
return { "base": self["base"].json(), "quote": self["quote"].json()
}
```

**keys**() → a set-like object providing a view on D's keys

**market**

Open the corresponding market.

**Returns** Instance of *bitshares.aio.market.Market* for the corresponding pair of assets.

**pop**(k[, d]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise *KeyError* is raised

**popitem**() → (k, v), remove and return some (key, value) pair as a

2-tuple; but raise *KeyError* if D is empty.

**classmethod set\_shared\_blockchain\_instance**(instance)

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters** *instance* (chaininstance) – Chain instance

**classmethod set\_shared\_config**(config)

This allows to set a config that will be used when calling *shared\_blockchain\_instance* and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance**()

This method allows to set the current instance as default

**setdefault**()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**symbols()**

**update** (`[E]`, `**F`) → None. Update `D` from dict/iterable `E` and `F`.

If `E` is present and has a `.keys()` method, then does: for `k` in `E`: `D[k] = E[k]` If `E` is present and lacks a `.keys()` method, then does: for `k, v` in `E`: `D[k] = v` In either case, this is followed by: for `k` in `F`: `D[k] = F[k]`

**values()** → an object providing a view on `D`'s values

**class** `bitshares.aio.price.Price(*args, **kwargs)`

Bases: `bitshares.aio.instance.BlockchainInstance`, `bitshares.aio.price.Price`

This class deals with all sorts of prices of any pair of assets to simplify dealing with the tuple:

```
(quote, base)

each being an instance of :class:`bitshares.amount.Amount`. The
amount themselves define the price.

.. note::

    The price (floating) is derived as ``base/quote``

:param list args: Allows to deal with different representations of a price
:param bitshares.aio.asset.Asset base: Base asset
:param bitshares.aio.asset.Asset quote: Quote asset
:param bitshares.aio.bitshares.BitShares blockchain_instance: BitShares instance
:returns: All data required to represent a price
:rtype: dict

Way to obtain a proper instance:

* ``args`` is a str with a price and two assets
* ``args`` can be a floating number and ``base`` and ``quote`` being
↳instances of :class:`bitshares.aio.asset.Asset`
* ``args`` can be a floating number and ``base`` and ``quote`` being
↳instances of ``str``
* ``args`` can be dict with keys ``price``, ``base``, and ``quote``
↳(*graphene balances*)
* ``args`` can be dict with keys ``base`` and ``quote``
* ``args`` can be dict with key ``receives`` (filled orders)
* ``args`` being a list of ``[quote, base]`` both being instances of
↳:class:`bitshares.aio.amount.Amount`
* ``args`` being a list of ``[quote, base]`` both being instances of ``str``
↳(``amount symbol``)
* ``base`` and ``quote`` being instances of :class:`bitshares.aio.asset.
↳Amount`

This allows instanciations like:

* ``Price("0.315 USD/BTS")``
* ``Price(0.315, base="USD", quote="BTS")``
* ``Price(0.315, base=Asset("USD"), quote=Asset("BTS"))``
* ``Price({"base": {"amount": 1, "asset_id": "1.3.0"}, "quote": {"amount": 10,
↳"asset_id": "1.3.106"}})``
* ``Price({"receives": {"amount": 1, "asset_id": "1.3.0"}, "pays": {"amount": 10,
↳"asset_id": "1.3.106"}}, base_asset=Asset("1.3.0"))``
```

(continues on next page)

(continued from previous page)

```
* ``Price(quote="10 GOLD", base="1 USD")``
* ``Price("10 GOLD", "1 USD")``
* ``Price(Amount("10 GOLD"), Amount("1 USD"))``
* ``Price(1.0, "USD/GOLD")``
```

Instances of this class can be used in regular mathematical expressions  
(``+-\*/%``) such as:

```
.. code-block:: python
```

```
>>> from bitshares.aio.price import Price
>>> await Price("0.3314 USD/BTS") * 2
0.662600000 USD/BTS
```

**as\_base** (*base*)

Returns the price instance so that the base asset is base.

Note: This makes a copy of the object!

**as\_quote** (*quote*)

Returns the price instance so that the quote asset is quote.

Note: This makes a copy of the object!

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of *bitshares.aio.instance.BlockchainInstance*

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject** (*cls*)

**invert** ()

Invert the price (e.g. go from USD/BTS into BTS/USD)

**items** () → a set-like object providing a view on D's items

**json** ()

```
return { "base": self["base"].json(), "quote": self["quote"].json()
}
```

**keys** () → a set-like object providing a view on D's keys

**market**

Open the corresponding market.

**Returns** Instance of `bitshares.aitmarket.Market` for the corresponding pair of assets.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise `KeyError` is raised

**popitem** () → (k, v), remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if D is empty.

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**symbols** ()

**update** ( $[E], **F$ ) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: `D[k] = E[k]` If E is present and lacks a `.keys()` method, then does: for k, v in E: `D[k] = v` In either case, this is followed by: for k in F: `D[k] = F[k]`

**values** () → an object providing a view on D's values

**class** `bitshares.aitprice.PriceFeed` (\*args, \*\*kwargs)

Bases: `bitshares.aitinstance.BlockchainInstance`, `bitshares.aitprice.PriceFeed`

This class is used to represent a price feed consisting of.

- a witness,
- a symbol,
- a core exchange rate,
- the maintenance collateral ratio,
- the max short squeeze ratio,
- a settlement price, and
- a date

**Parameters** *blockchain\_instance* (`bitshares.aitbitshares.BitShares`) – BitShares instance

**bitshares**

Alias for the specific blockchain.

**blockchain****blockchain\_instance\_class**

alias of `bitshares.aio.instance.BlockchainInstance`

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. `bitshares.BitShares`

**classmethod inject** (cls)

**items** () → a set-like object providing a view on D's items

**keys** () → a set-like object providing a view on D's keys

**pop** (k[, d]) → v, remove specified key and return the corresponding value.  
If key is not found, d is returned if given, otherwise `KeyError` is raised

**popitem** () → (k, v), remove and return some (key, value) pair as a  
2-tuple; but raise `KeyError` if D is empty.

**classmethod set\_shared\_blockchain\_instance** (instance)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** **instance** (chaininstance) – Chain instance

**classmethod set\_shared\_config** (config)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**update** ([E], \*\*F) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values



**class** `bitshares.aitprice.UpdateCallOrder` (*call*, *\*\*kwargs*)

Bases: `bitshares.aitprice.Price`

This class inherits `bitshares.aitprice.Price` but has the base and quote Amounts not only be used to represent the **call price** (as a ratio of base and quote).

**Parameters** `blockchain_instance` (`bitshares.aitbitshares.BitShares`) – Bit-Shares instance

**as\_base** (*base*)

Returns the price instance so that the base asset is base.

Note: This makes a copy of the object!

**as\_quote** (*quote*)

Returns the price instance so that the quote asset is quote.

Note: This makes a copy of the object!

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.aitinstance.BlockchainInstance`

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. `bitshares.BitShares`

**classmethod inject** (*cls*)

**invert** ()

Invert the price (e.g. go from USD/BTS into BTS/USD)

**items** () → a set-like object providing a view on D's items

**json** ()

```
return { "base": self["base"].json(), "quote": self["quote"].json()
}
```

**keys** () → a set-like object providing a view on D's keys

**market**

Open the corresponding market.

**Returns** Instance of `bitshares.aitmarket.Market` for the corresponding pair of assets.

**pop** (*k*, *d*) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a

2-tuple; but raise `KeyError` if *D* is empty.

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**symbols** ()

**update** ([*E*], *\*\*F*) → `None`. Update *D* from dict/iterable *E* and *F*.

If *E* is present and has a `.keys()` method, then does: for *k* in *E*: *D*[*k*] = *E*[*k*] If *E* is present and lacks a `.keys()` method, then does: for *k*, *v* in *E*: *D*[*k*] = *v* In either case, this is followed by: for *k* in *F*: *D*[*k*] = *F*[*k*]

**values** () → an object providing a view on *D*'s values

## bitshares.aid.proposal module

**class** `bitshares.aid.proposal.Proposal` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.aid.instance.BlockchainInstance`, `bitshares.aid.proposal.Proposal`

Read data about a Proposal Balance in the chain.

**Parameters**

- **id** (*str*) – Id of the proposal
- **blockchain\_instance** (*bitshares*) – `BitShares()` instance to use when accessing a RPC

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.aid.instance.BlockchainInstance`

**classmethod cache\_object** (*data*, *key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear()** → None. Remove all items from D.

**classmethod clear\_cache()**  
Clear/Reset the entire Cache

**copy()** → a shallow copy of D

**define\_classes()**  
Needs to define instance variables that provide classes

**expiration**

**fromkeys()**  
Create a new dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**get\_instance\_class()**  
Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache(id)**  
Get an element from the cache explicitly

**identifier = None**

**incached(id)**  
Is an element cached?

**classmethod inject(cls)**

**is\_in\_review**

**items()**  
This overrides items() so that refresh() is called if the object is not already fetched

**keys()** → a set-like object providing a view on D's keys

**static objectid\_valid(i)**  
Test if a string looks like a regular object id of the form::

```
xxxxx.yyyyyy.zzzz
```

with those being numbers.

**perform\_id\_tests = True**

**pop(k[, d])** → v, remove specified key and return the corresponding value.  
If key is not found, d is returned if given, otherwise KeyError is raised

**popitem()** → (k, v), remove and return some (key, value) pair as a 2-tuple; but raise KeyError if D is empty.

**proposed\_operations**

**proposer**  
Return the proposer of the proposal if available in the backend, else returns None

**refresh()**

**review\_period**

**static set\_cache\_store(klass, \*args, \*\*kwargs)**

**classmethod set\_shared\_blockchain\_instance(instance)**  
This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters** `instance` (*chaininstance*) – Chain instance

**classmethod** `set_shared_config` (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (*data*, *key*='id')

Cache the list

**Parameters** `data` (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)

Alias for `objectid_valid`

**testid** (*id*)

In contrast to validity, this method tests if the objectid matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id** = None

**type\_ids** = []

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** `bitshares.aio.proposal.Proposals` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.aio.instance.BlockchainInstance`, `bitshares.aio.proposal.Proposals`

Obtain a list of pending proposals for an account.

**Parameters**

- **account** (*str*) – Account name
- **blockchain\_instance** (*bitshares*) – BitShares() instance to use when accessing a RPC

**append** ()

Append object to the end of the list.

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.aio.instance.BlockchainInstance`

**cache** (*key*)  
(legacy) store the current object with key *key*.

**classmethod cache\_objects** (*data, key=None*)  
This classmethod allows to feed multiple objects into the cache is is mostly used for testing

**chain**  
Short form for blockchain (for the lazy)

**clear** ()  
Remove all items from list.

**classmethod clear\_cache** ()  
Clear/Reset the entire Cache

**copy** ()  
Return a shallow copy of the list.

**count** ()  
Return number of occurrences of value.

**define\_classes** ()  
Needs to define instance variables that provide classes

**extend** ()  
Extend list by appending elements from the iterable.

**get\_instance\_class** ()  
Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)  
Get an element from the cache explicitly

**identifier = None**

**incached** (*id*)  
Is an element cached?

**index** ()  
Return first index of value.  
  
Raises *ValueError* if the value is not present.

**classmethod inject** (*cls*)

**insert** ()  
Insert object before index.

**items** ()  
This overrides items() so that refresh() is called if the object is not already fetched

**pop** ()  
Remove and return item at index (default last).  
  
Raises *IndexError* if list is empty or index is out of range.

**refresh** (*\*args, \*\*kwargs*)  
Interface that needs to be implemented. This method is called when an object is requested that has not yet been fetched/stored

**remove** ()  
Remove first occurrence of value.  
  
Raises *ValueError* if the value is not present.

**reverse()**

Reverse *IN PLACE*.

**static set\_cache\_store** (*klass*, \**args*, \*\**kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**sort()**

Stable sort *IN PLACE*.

**store** (*data*, *key=None*, \**args*, \*\**kwargs*)

Cache the list

**Parameters** *data* (*list*) – List of objects to cache

## bitshares.aio.transactionbuilder module

**class** `bitshares.aio.transactionbuilder.ProposalBuilder` (\**args*, \*\**kwargs*)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.aio.transactionbuilder.ProposalBuilder`

Proposal Builder allows us to construct an independent Proposal that may later be added to an instance of TransactionBuilder.

### Parameters

- **proposer** (*str*) – Account name of the proposing user
- **proposal\_expiration** (*int*) – Number seconds until the proposal is supposed to expire
- **proposal\_review** (*int*) – Number of seconds for review of the proposal
- **transactionbuilder.TransactionBuilder** – Specify your own instance of transaction builder (optional)
- **blockchain\_instance** (*instance*) – Blockchain instance

**appendOps** (*ops*, *append\_to=None*)

Append op(s) to the transaction builder

**Parameters** *ops* (*list*) – One or a list of operations

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**broadcast()****chain**

Short form for blockchain (for the lazy)

**define\_classes()**

Needs to define instance variables that provide classes

**get\_instance\_class()**

Should return the Chain instance class, e.g. `bitshares.BitShares`

**get\_parent()**

This allows to refer to the actual parent of the Proposal

**get\_raw()**

Returns an instance of base “Operations” for further processing

**classmethod inject(cls)****is\_empty()****json()**

Return the json formatted version of this proposal

**list\_operations()****set\_expiration(p)****set\_parent(p)****set\_proposer(p)****set\_review(p)****classmethod set\_shared\_blockchain\_instance(instance)**

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** `instance(chaininstance)` – Chain instance

**classmethod set\_shared\_config(config)**

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**class** `bitshares.aio.transactionbuilder.TransactionBuilder(*args, **kwargs)`

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.aio.transactionbuilder.TransactionBuilder`

This class simplifies the creation of transactions by adding operations and signers.

**addSigningInformation(account, permission)**

This is a private method that adds side information to a unsigned/partial transaction in order to simplify later signing (e.g. for multisig or coldstorage)

FIXME: Does not work with owner keys!

**add\_required\_fees** (*ops*, *asset\_id*='1.3.0')

Auxiliary method to obtain the required fees for a set of operations. Requires a websocket connection to a witness node!

**appendMissingSignatures** ()

Store which accounts/keys are supposed to sign the transaction

This method is used for an offline-signer!

**appendOps** (*ops*, *append\_to*=None)

Append op(s) to the transaction builder

**Parameters** **ops** (*list*) – One or a list of operations

**appendSigner** (*accounts*, *permission*)

Try to obtain the wif key from the wallet by telling which account and permission is supposed to sign the transaction

**appendWif** (*wif*)

Add a wif that should be used for signing of the transaction.

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of *bitshares.instance.BlockchainInstance*

**broadcast** ()

Broadcast a transaction to the blockchain network

**Parameters** **tx** (*tx*) – Signed transaction to broadcast

**chain**

Short form for blockchain (for the lazy)

**clear** ()

Clear the transaction builder and start from scratch

**constructTx** ()

Construct the actual transaction and store it in the class's dict store

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_block\_params** (*use\_head\_block*=False)

Auxiliary method to obtain *ref\_block\_num* and *ref\_block\_prefix*. Requires a websocket connection to a witness node!

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**get\_parent** ()

TransactionBuilders don't have parents, they are their own parent

**classmethod inject** (*cls*)



**is\_empty()**

**items()** → a set-like object providing a view on D's items

**json()**

Show the transaction as plain json

**keys()** → a set-like object providing a view on D's keys

**list\_operations()**

**permission\_types** = ['active', 'owner']

**pop**(*k*, *d*) → *v*, remove specified key and return the corresponding value.  
If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem()** → (*k*, *v*), remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if D is empty.

**set\_expiration**(*p*)

**set\_fee\_asset**(*fee\_asset*)

Set asset to fee

**classmethod set\_shared\_blockchain\_instance**(*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance*(*chaininstance*) – Chain instance

**classmethod set\_shared\_config**(*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**sign()**

Sign a provided transaction with the provided key(s)

**Parameters**

- **tx**(*dict*) – The transaction to be signed and returned
- **wifs**(*string*) – One or many wif keys to use for signing a transaction. If not present, the keys will be loaded from the wallet as defined in “missing\_signatures” key of the transactions.

**update**(*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for *k* in E: *D*[*k*] = *E*[*k*] If E is present and lacks a `.keys()` method, then does: for *k*, *v* in E: *D*[*k*] = *v* In either case, this is followed by: for *k* in F: *D*[*k*] = *F*[*k*]

**values()** → an object providing a view on D's values

**verify\_authority()**

Verify the authority of the signed transaction

**bitshares.aio.vesting module****class** bitshares.aio.vesting.Vesting(\*args, \*\*kwargs)Bases: *bitshares.aio.instance.BlockchainInstance*, *bitshares.aio.vesting.Vesting*

Read data about a Vesting Balance in the chain.

**Parameters**

- **id**(*str*) – Id of the vesting balance
- **blockchain\_instance** (*bitshares*) – BitShares() instance to use when accessing a RPC

**account****bitshares**

Alias for the specific blockchain.

**blockchain****blockchain\_instance\_class**alias of *bitshares.aio.instance.BlockchainInstance***classmethod** **cache\_object** (*data*, *key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**claim** (*amount=None*)**claimable****clear** () → None. Remove all items from D.**classmethod** **clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()Should return the Chain instance class, e.g. *bitshares.BitShares***getfromcache** (*id*)

Get an element from the cache explicitly

**identifier** = None**incached** (*id*)

Is an element cached?

**classmethod** **inject** (*cls*)**items** ()

This overrides items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static objectid\_valid** (i)

Test if a string looks like a regular object id of the form::

```
xxxx.yyyyy.zzzz
```

with those being numbers.

**perform\_id\_tests** = True

**pop** (k[, d]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

**popitem** () → (k, v), remove and return some (key, value) pair as a

2-tuple; but raise KeyError if D is empty.

**refresh** ()

**static set\_cache\_store** (klass, \*args, \*\*kwargs)

**classmethod set\_shared\_blockchain\_instance** (instance)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (config)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (data, key='id')

Cache the list

**Parameters** *data* (*list*) – List of objects to cache

**test\_valid\_objectid** (i)

Alias for `objectid_valid`

**testid** (id)

In contrast to validity, this method tests if the objectid matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id** = None

**type\_ids** = []

**update** ([E], \*\*F) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**bitshares.aio.wallet module**

```
class bitshares.aio.wallet.Wallet (*args, **kwargs)
    Bases: bitshares.instance.BlockchainInstance, bitshares.aio.wallet.Wallet

addPrivateKey (wif)
    Add a private key to the wallet database

bitshares
    Alias for the specific blockchain.

blockchain

blockchain_instance_class
    alias of bitshares.instance.BlockchainInstance

chain
    Short form for blockchain (for the lazy)

changePassphrase (new_pwd)
    Change the passphrase for the wallet database

create (pwd)
    Alias for newWallet()

created ()
    Do we have a wallet database already?

define_classes ()
    Needs to define instance variables that provide classes

getAccountFromPrivateKey (wif)
    Obtain account name from private key

getAccountFromPublicKey (pub)
    Obtain the first account name from public key

getAccounts ()
    Return all accounts installed in the wallet database

getAccountsFromPublicKey (pub)
    Obtain all accounts associated with a public key

getActiveKeyForAccount (name)
    Obtain owner Active Key for an account from the wallet database

getAllAccounts (pub)
    Get the account data for a public key (all accounts found for this public key)

getKeyType (account, pub)
    Get key type

getMemoKeyForAccount (name)
    Obtain owner Memo Key for an account from the wallet database

getOwnerKeyForAccount (name)
    Obtain owner Private Key for an account from the wallet database

getPrivateKeyForPublicKey (pub)
    Obtain the private key for a given public key

    Parameters pub (str) – Public Key
```

**getPublicKeys** (*current=False*)

Return all installed public keys

**Parameters** **current** (*bool*) – If true, returns only keys for currently connected blockchain

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject** (*cls*)

**is\_encrypted** ()

Is the key store encrypted?

**lock** ()

Lock the wallet database

**locked** ()

Is the wallet database locked?

**newWallet** (*pwd*)

Create a new wallet database

**prefix**

**privatekey** (*key*)

**publickey\_from\_wif** (*wif*)

**removeAccount** (*account*)

Remove all keys associated with a given account

**removePrivateKeyFromPublicKey** (*pub*)

Remove a key from the wallet database

**rpc**

**setKeys** (*loadkeys*)

This method is strictly only for in memory keys that are passed to Wallet with the *keys* argument

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters** **instance** (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling *shared\_blockchain\_instance* and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**shared\_blockchain\_instance** ()

This method will initialize *SharedInstance.instance* and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**unlock** (*pwd*)

Unlock the wallet database

**unlocked** ()

Is the wallet database unlocked?

**wipe** (*sure=False*)

**bitshares.aio.witness module****class** bitshares.aio.witness.**Witness** (\*args, \*\*kwargs)Bases: *bitshares.aio.instance.BlockchainInstance*, *bitshares.aio.witness.Witness*

Read data about a witness in the chain.

**Parameters**

- **account\_name** (*str*) – Name of the witness
- **blockchain\_instance** (*bitshares*) – BitShares() instance to use when accessing a RPC

**account****bitshares**

Alias for the specific blockchain.

**blockchain****blockchain\_instance\_class**alias of *bitshares.aio.instance.BlockchainInstance***classmethod** **cache\_object** (*data*, *key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.**classmethod** **clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()Should return the Chain instance class, e.g. *bitshares.BitShares***getfromcache** (*id*)

Get an element from the cache explicitly

**identifier** = None**incached** (*id*)

Is an element cached?

**classmethod** **inject** (*cls*)**is\_active****items** ()

This overrides items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static objectid\_valid(*i*)**

Test if a string looks like a regular object id of the form::

```
xxxx.yyzzz.zzzz
```

with those being numbers.

**perform\_id\_tests = True**

**pop(*k*, *d*)** → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem()** → (*k*, *v*), remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if *D* is empty.

**refresh()**

**static set\_cache\_store(*klass*, \**args*, \*\**kwargs*)**

**classmethod set\_shared\_blockchain\_instance(*instance*)**

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters *instance*** (*chaininstance*) – Chain instance

**classmethod set\_shared\_config(*config*)**

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id = 1**

**store(*data*, *key*='id')**

Cache the list

**Parameters *data*** (*list*) – List of objects to cache

**test\_valid\_objectid(*i*)**

Alias for `objectid_valid`

**testid(*id*)**

In contrast to validity, this method tests if the objectid matches the type\_id provided in `self.type_id` or `self.type_ids`

**type\_id = None**

**type\_ids = []**

**update(*[E]*, \*\**F*)** → `None`. Update *D* from dict/iterable *E* and *F*.

If *E* is present and has a `.keys()` method, then does: for *k* in *E*: *D*[*k*] = *E*[*k*] If *E* is present and lacks a `.keys()` method, then does: for *k*, *v* in *E*: *D*[*k*] = *v* In either case, this is followed by: for *k* in *F*: *D*[*k*] = *F*[*k*]

**values()** → an object providing a view on *D*'s values

**weight**

```
class bitshares.aio.witness.Witnesses(*args, **kwargs)
    Bases: bitshares.aio.instance.BlockchainInstance, bitshares.aio.witness.Witnesses
```

Obtain a list of **active** witnesses and the current schedule.

#### Parameters

- **only\_active** (*bool*) – (False) Only return witnesses that are actively producing blocks
- **blockchain\_instance** (*bitshares*) – BitShares() instance to use when accessing a RPC

```
append()
```

Append object to the end of the list.

```
bitshares
```

Alias for the specific blockchain.

```
blockchain
```

```
blockchain_instance_class
```

alias of *bitshares.aio.instance.BlockchainInstance*

```
cache(key)
```

(legacy) store the current object with key *key*.

```
classmethod cache_objects(data, key=None)
```

This classmethod allows to feed multiple objects into the cache is is mostly used for testing

```
chain
```

Short form for blockchain (for the lazy)

```
clear()
```

Remove all items from list.

```
classmethod clear_cache()
```

Clear/Reset the entire Cache

```
copy()
```

Return a shallow copy of the list.

```
count()
```

Return number of occurrences of value.

```
define_classes()
```

Needs to define instance variables that provide classes

```
extend()
```

Extend list by appending elements from the iterable.

```
get_instance_class()
```

Should return the Chain instance class, e.g. *bitshares.BitShares*

```
getfromcache(id)
```

Get an element from the cache explicitly

```
identifier = None
```

```
incached(id)
```

Is an element cached?

```
index()
```

Return first index of value.



Raises `ValueError` if the value is not present.

**classmethod inject** (*cls*)

**insert** ()  
Insert object before index.

**items** ()  
This overrides `items()` so that `refresh()` is called if the object is not already fetched

**pop** ()  
Remove and return item at index (default last).  
  
Raises `IndexError` if list is empty or index is out of range.

**refresh** (\*args, \*\*kwargs)  
Interface that needs to be implemented. This method is called when an object is requested that has not yet been fetched/stored

**remove** ()  
Remove first occurrence of value.  
  
Raises `ValueError` if the value is not present.

**reverse** ()  
Reverse *IN PLACE*.

**static set\_cache\_store** (klass, \*args, \*\*kwargs)

**classmethod set\_shared\_blockchain\_instance** (instance)  
This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters instance** (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (config)  
This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()  
This method allows to set the current instance as default

**shared\_blockchain\_instance** ()  
This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**sort** ()  
Stable sort *IN PLACE*.

**store** (data, key=None, \*args, \*\*kwargs)  
Cache the list

**Parameters data** (*list*) – List of objects to cache

## bitshares.aid.worker module

**class** bitshares.aid.worker.**Worker** (\*args, \*\*kwargs)  
Bases: `bitshares.aid.instance.BlockchainInstance`, `bitshares.aid.worker.Worker`  
Read data about a worker in the chain.

**Parameters**

- **id** (*str*) – id of the worker

- **blockchain\_instance** (*bitshares*) – BitShares() instance to use when accessing a RPC

**account**

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of *bitshares.aio.instance.BlockchainInstance*

**classmethod cache\_object** (*data, key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier = None**

**incached** (*id*)

Is an element cached?

**classmethod inject** (*cls*)

**items** ()

This overrides items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static objectid\_valid** (*i*)

Test if a string looks like a regular object id of the form::

`xxxx.yyzzz.zzzz`

with those being numbers.

**perform\_id\_tests = True**

**pop** (*k[, d]*) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

**popitem()** → (k, v), remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if `D` is empty.

**post\_format()**

**refresh()**

**static set\_cache\_store** (*klass*, \**args*, \*\**kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id = 1**

**store** (*data*, *key*='id')

Cache the list

**Parameters** *data* (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)

Alias for `objectid_valid`

**testid** (*id*)

In contrast to validity, this method tests if the `objectid` matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id = None**

**type\_ids = []**

**update** (*[E]*, \*\**F*) → None. Update `D` from dict/iterable `E` and `F`.

If `E` is present and has a `.keys()` method, then does: for `k` in `E`: `D[k] = E[k]` If `E` is present and lacks a `.keys()` method, then does: for `k, v` in `E`: `D[k] = v` In either case, this is followed by: for `k` in `F`: `D[k] = F[k]`

**values()** → an object providing a view on `D`'s values

**class** `bitshares.aio.worker.Workers` (\**args*, \*\**kwargs*)

Bases: `bitshares.aio.instance.BlockchainInstance`, `bitshares.aio.worker.Workers`

Obtain a list of workers for an account.

**Parameters**

- **account\_name/id** (*str*) – Name/id of the account (optional)
- **blockchain\_instance** (*bitshares*) – `BitShares()` instance to use when accessing a RPC

**append()**  
Append object to the end of the list.

**bitshares**  
Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**  
alias of *bitshares.aio.instance.BlockchainInstance*

**cache(key)**  
(legacy) store the current object with key *key*.

**classmethod cache\_objects(data, key=None)**  
This classmethod allows to feed multiple objects into the cache is is mostly used for testing

**chain**  
Short form for blockchain (for the lazy)

**clear()**  
Remove all items from list.

**classmethod clear\_cache()**  
Clear/Reset the entire Cache

**copy()**  
Return a shallow copy of the list.

**count()**  
Return number of occurrences of value.

**define\_classes()**  
Needs to define instance variables that provide classes

**extend()**  
Extend list by appending elements from the iterable.

**get\_instance\_class()**  
Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache(id)**  
Get an element from the cache explicitly

**identifier = None**

**incached(id)**  
Is an element cached?

**index()**  
Return first index of value.  
  
Raises *ValueError* if the value is not present.

**classmethod inject(cls)**

**insert()**  
Insert object before index.

**items()**  
This overrides items() so that refresh() is called if the object is not already fetched

**pop()**  
Remove and return item at index (default last).

Raises `IndexError` if list is empty or index is out of range.

**refresh** (\*args, \*\*kwargs)

Interface that needs to be implemented. This method is called when an object is requested that has not yet been fetched/stored

**remove** ()

Remove first occurrence of value.

Raises `ValueError` if the value is not present.

**reverse** ()

Reverse *IN PLACE*.

**static set\_cache\_store** (klass, \*args, \*\*kwargs)

**classmethod set\_shared\_blockchain\_instance** (instance)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** **instance** (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (config)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**sort** ()

Stable sort *IN PLACE*.

**store** (data, key=None, \*args, \*\*kwargs)

Cache the list

**Parameters** **data** (*list*) – List of objects to cache

## Module contents

### Submodules

#### bitshares.account module

**class** `bitshares.account.Account` (\*args, \*\*kwargs)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.account.Account`

This class allows to easily access Account data.

#### Parameters

- **account\_name** (*str*) – Name of the account
- **blockchain\_instance** (`bitshares.bitshares.BitShares`) – BitShares instance
- **full** (*bool*) – Obtain all account data including orders, positions, etc.
- **lazy** (*bool*) – Use lazy loading
- **full** – Obtain all account data including orders, positions, etc.

**Returns** Account data

**Return type** dictionary

**Raises** `bitshares.exceptions.AccountDoesNotExistException` – if account does not exist

Instances of this class are dictionaries that come with additional methods (see below) that allow dealing with an account and it's corresponding functions.

```
from bitshares.account import Account
account = Account("init0")
print(account)
```

---

**Note:** This class comes with its own caching function to reduce the load on the API server. Instances of this class can be refreshed with `Account.refresh()`.

---

**balance** (*symbol*)

Obtain the balance of a specific Asset. This call returns instances of `amount.Amount`.

**balances**

List balances of an account. This call returns instances of `amount.Amount`.

**bitshares**

Alias for the specific blockchain.

**blacklist** (*account*)

Add an other account to the blacklist of this account

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**classmethod cache\_object** (*data, key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**call\_positions**

Alias for `:func:bitshares.account.Account.callpositions`.

**callpositions**

List call positions (collateralized positions *Market Pegged Assets*)

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**ensure\_full** ()

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**history** (*first=0, last=0, limit=-1, only\_ops=[], exclude\_ops=[]*)

Returns a generator for individual account transactions. The latest operation will be first. This call can be used in a `for` loop.

#### Parameters

- **first** (*int*) – sequence number of the first transaction to return (*optional*)
- **last** (*int*) – sequence number of the last transaction to return (*optional*)
- **limit** (*int*) – limit number of transactions to return (*optional*)
- **only\_ops** (*array*) – Limit generator by these operations (*optional*)
- **exclude\_ops** (*array*) – Exclude these operations from generator (*optional*).

... **note::** `only_ops` and `exclude_ops` takes an array of strings: The full list of operation ID's can be found in `operationids.py`. Example: `['transfer', 'fill_order']`

**identifier** = `None`

**incached** (*id*)

Is an element cached?

**classmethod inject** (*cls*)

**is\_fully\_loaded**

Is this instance fully loaded / e.g. all data available?

**is\_ltm**

Is the account a lifetime member (LTM)?

**items** ()

This overwrites `items()` so that `refresh()` is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**name**

**nolist** (*account*)

Remove an other account from any list of this account

**static objectid\_valid** (*i*)

Test if a string looks like a regular object id of the form::

```
xxxx.yyzz.zzzz
```

with those being numbers.

**openorders**

Returns open Orders.

**perform\_id\_tests** = `True`

**pop** (*k*, *d*) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem()** → (k, v), remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if `D` is empty.

**refresh()**  
Refresh/Obtain an account's data from the API server

**static set\_cache\_store** (*klass*, \**args*, \*\**kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)  
This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)  
This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**  
This method allows to set the current instance as default

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
  
Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**  
This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id = 1**

**store** (*data*, *key*='id')  
Cache the list

**Parameters** *data* (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)  
Alias for `objectid_valid`

**testid** (*id*)  
In contrast to validity, this method tests if the `objectid` matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id = None**

**type\_ids = []**

**update** ([*E*], \*\**F*) → None. Update `D` from dict/iterable `E` and `F`.  
If `E` is present and has a `.keys()` method, then does: for `k` in `E`: `D[k] = E[k]` If `E` is present and lacks a `.keys()` method, then does: for `k`, `v` in `E`: `D[k] = v` In either case, this is followed by: for `k` in `F`: `D[k] = F[k]`

**upgrade()**  
Upgrade account to life time member

**values()** → an object providing a view on `D`'s values

**whitelist** (*account*)  
Add an other account to the whitelist of this account

**class** `bitshares.account.AccountUpdate` (\**args*, \*\**kwargs*)  
Bases: `bitshares.instance.BlockchainInstance`, `bitshares.account.AccountUpdate`  
  
This purpose of this class is to keep track of account updates as they are pushed through by `bitshares.notify.Notify`.



Instances of this class are dictionaries and take the following form:

**account**

In order to obtain the actual `account.Account` from this class, you can use the `account` attribute.

**bitshares**

Alias for the specific blockchain.

**blockchain****blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. `bitshares.BitShares`

**classmethod inject** (cls)

**items** () → a set-like object providing a view on D's items

**keys** () → a set-like object providing a view on D's keys

**pop** (k[, d]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise `KeyError` is raised

**popitem** () → (k, v), remove and return some (key, value) pair as a

2-tuple; but raise `KeyError` if D is empty.

**classmethod set\_shared\_blockchain\_instance** (instance)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** `instance` (chaininstance) – Chain instance

**classmethod set\_shared\_config** (config)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**update** (*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

## bitshares.amount module

**class** `bitshares.amount.Amount` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.amount.Amount`

This class deals with Amounts of any asset to simplify dealing with the tuple:

`(amount, asset)`

### Parameters

- **args** (*list*) – Allows to deal with different representations of an amount
- **amount** (*float*) – Let's create an instance with a specific amount
- **asset** (*str*) – Let's you create an instance with a specific asset (symbol)
- **blockchain\_instance** (`bitshares.bitshares.BitShares`) – BitShares instance

**Returns** All data required to represent an Amount/Asset

**Return type** dict

**Raises** **ValueError** – if the data provided is not recognized

```
from peerplays.amount import Amount
from peerplays.asset import Asset
a = Amount("1 USD")
b = Amount(1, "USD")
c = Amount("20", Asset("USD"))
a + b
a * 2
a += b
a /= 2.0
```

Way to obtain a proper instance:

- args can be a string, e.g.: "1 USD"
- args can be a dictionary containing amount and asset\_id
- args can be a dictionary containing amount and asset
- args can be a list of a float and str (symbol)
- args can be a list of a float and a `bitshares.asset.Asset`
- amount and asset are defined manually

An instance is a dictionary and comes with the following keys:

- amount (float)
- symbol (str)
- asset (instance of `bitshares.asset.Asset`)

Instances of this class can be used in regular mathematical expressions (+-\*/%) such as:

```
Amount("1 USD") * 2
Amount("15 GOLD") + Amount("0.5 GOLD")
```

**amount**

Returns the amount as float

**asset**

Returns the asset as instance of *asset.Asset*

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of *bitshares.instance.BlockchainInstance*

**chain**

Short form for blockchain (for the lazy)

**clear()** → None. Remove all items from D.

**copy()**

Copy the instance and make sure not to use a reference

**define\_classes()**

Needs to define instance variables that provide classes

**fromkeys()**

Create a new dictionary with keys from iterable and values set to value.

**get()**

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class()**

Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject(cls)**

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**pop(k[, d])** → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise *KeyError* is raised

**popitem()** → (k, v), remove and return some (key, value) pair as a

2-tuple; but raise *KeyError* if D is empty.

**classmethod set\_shared\_blockchain\_instance(instance)**

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config(config)**

This allows to set a config that will be used when calling *shared\_blockchain\_instance* and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**symbol**

Returns the symbol of the asset

**tuple** ()

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

## bitshares.asset module

**class** `bitshares.asset.Asset` (\*args, \*\*kwargs)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.asset.Asset`

Deals with Assets of the network.

### Parameters

- **Asset** (*str*) – Symbol name or object id of an asset
- **lazy** (*bool*) – Lazy loading
- **full** (*bool*) – Also obtain bitasset-data and dynamic asset data
- **blockchain\_instance** (`bitshares.bitshares.BitShares`) – BitShares instance

**Returns** All data of an asset

**Return type** dict

---

**Note:** This class comes with its own caching function to reduce the load on the API server. Instances of this class can be refreshed with `Asset.refresh()`.

---

**add\_authorities** (*type*, *authorities=None*)

Add authorities to an assets white/black list.

### Parameters

- **type** (*str*) – blacklist or whitelist
- **authorities** (*list*) – List of authorities (Accounts)

**add\_markets** (*type*, *authorities=None*, *force\_enable=True*)

Add markets to an assets white/black list.

### Parameters

- **type** (*str*) – blacklist or whitelist
- **markets** (*list*) – List of markets (assets)

- **force\_enable** (*bool*) – Force enable `white_list` flag

**bitshares**

Alias for the specific blockchain.

**blockchain****blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**classmethod cache\_object** (*data, key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**calls****chain**

Short form for blockchain (for the lazy)

**change\_issuer** (*new\_issuer, \*\*kwargs*)

Change asset issuer (needs signing with owner key!)

**Parameters** **new\_issuer** (*str*) – account name

**clear** () → None. Remove all items from D.**classmethod clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D**define\_classes** ()

Needs to define instance variables that provide classes

**disableflag** (*flag*)

Enable a certain flag.

**Parameters** **flag** (*str*) – Flag name

**enableflag** (*flag*)

Enable a certain flag.

**Parameters** **flag** (*str*) – Flag name

**ensure\_full** ()**feed****feeds****flags**

List the permissions that are currently used (flags)

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_call\_orders** (*limit=100*)**get\_instance\_class** ()

Should return the Chain instance class, e.g. `bitshares.BitShares`

**get\_settle\_orders** (*limit=100*)**getfromcache** (*id*)

Get an element from the cache explicitly

**halt()**

Halt this asset from being moved or traded.

**identifier = None**

**incached(*id*)**

Is an element cached?

**classmethod inject(*cls*)**

**is\_bitasset**

Is the asset a market pegged asset?

**is\_fully\_loaded**

Is this instance fully loaded / e.g. all data available?

**issue(*amount*, *to*, *memo*=None, *\*\*kwargs*)**

Issue new shares of an asset.

#### Parameters

- **amount** (*float*) – Amount to issue
- **to** (*str*) – Recipient
- **memo** (*str*) – (optional) Memo message

**items()**

This overwrites items() so that refresh() is called if the object is not already fetched

**keys()** → a set-like object providing a view on D's keys

**market\_fee\_percent**

**max\_market\_fee**

**static objectid\_valid(*i*)**

Test if a string looks like a regular object id of the form::

`xxxx.yyzzz.zzzz`

with those being numbers.

**perform\_id\_tests = True**

**permissions**

List the permissions for this asset that the issuer can obtain

**pop(*k*[, *d*])** → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem()** → (*k*, *v*), remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if *D* is empty.

**precision**

**refresh()**

Refresh the data from the API server

**release(*whitelist\_authorities*=None, *blacklist\_authorities*=None, *whitelist\_markets*=None, *blacklist\_markets*=None)**

Release this asset and allow unrestricted transfer, trading, etc.

#### Parameters

- **whitelist\_authorities** (*list*) – List of accounts that serve as whitelist authorities
- **blacklist\_authorities** (*list*) – List of accounts that serve as blacklist authorities
- **whitelist\_markets** (*list*) – List of assets to allow trading with
- **blacklist\_markets** (*list*) – List of assets to prevent trading with

**remove\_authorities** (*type, authorities=None*)

Remove authorities from an assets white/black list.

#### Parameters

- **type** (*str*) – blacklist or whitelist
- **authorities** (*list*) – List of authorities (Accounts)

**remove\_markets** (*type, authorities=None*)

Remove markets from an assets white/black list.

#### Parameters

- **type** (*str*) – blacklist or whitelist
- **markets** (*list*) – List of markets (assets)

**seize** (*from\_account, to\_account, amount, \*\*kwargs*)

Seize amount from an account and send to another.

... note:: This requires the **override\_authority** to be set for this asset!

#### Parameters

- **from\_account** (`bitshares.account.Account`) – From this account
- **to\_account** (`bitshares.account.Account`) – To this account
- **amount** (`bitshares.amount.Amount`) – Amount to seize

**static set\_cache\_store** (*klass, \*args, \*\*kwargs*)

**set\_market\_fee** (*percentage\_fee, max\_market\_fee, \*\*kwargs*)

Set trading percentage fee.

#### Parameters

- **percentage\_fee** (*float*) – Percentage of fee
- **max\_market\_fee** (`bitshares.amount.Amount`) – Max Fee

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** **instance** (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**setoptions** (*flags*, *\*\*kwargs*)

Enable a certain flag.

Flags:

- charge\_market\_fee
- white\_list
- override\_authority
- transfer\_restricted
- disable\_force\_settle
- global\_settle
- disable\_confidential
- witness\_fed\_asset
- committee\_fed\_asset

**Parameters** **flag** (*dict*) – dictionary of flags and boolean

**settlements**

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (*data*, *key='id'*)

Cache the list

**Parameters** **data** (*list*) – List of objects to cache

**symbol**

**test\_valid\_objectid** (*i*)

Alias for objectid\_valid

**testid** (*id*)

In contrast to validity, this method tests if the objectid matches the type\_id provided in `self.type_id` or `self.type_ids`

**type\_id** = None

**type\_ids** = []

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**update\_cer** (*cer*, *account=None*, *\*\*kwargs*)

Update the Core Exchange Rate (CER) of an asset

**update\_feed\_producers** (*producers*)

Update bitasset feed producers.



**Parameters** `producers` (*list*) – List of accounts that are allowed to produce a feed  
**values** () → an object providing a view on D’s values

## bitshares.bitshares module

**class** `bitshares.bitshares.BitShares` (*node=""*, *rpcuser=""*, *rpcpassword=""*, *debug=False*,  
*skip\_wallet\_init=False*, *\*\*kwargs*)  
 Bases: `graphenecommon.chain.AbstractGrapheneChain`

Connect to the BitShares network.

### Parameters

- **node** (*str*) – Node to connect to (*optional*)
- **rpcuser** (*str*) – RPC user (*optional*)
- **rpcpassword** (*str*) – RPC password (*optional*)
- **nobroadcast** (*bool*) – Do **not** broadcast a transaction! (*optional*)
- **debug** (*bool*) – Enable Debugging (*optional*)
- **keys** (*array, dict, string*) – Predefine the wif keys to shortcut the wallet database (*optional*)
- **offline** (*bool*) – Boolean to prevent connecting to network (defaults to `False`) (*optional*)
- **proposer** (*str*) – Propose a transaction using this proposer (*optional*)
- **proposal\_expiration** (*int*) – Expiration time (in seconds) for the proposal (*optional*)
- **proposal\_review** (*int*) – Review period (in seconds) for the proposal (*optional*)
- **expiration** (*int*) – Delay in seconds until transactions are supposed to expire (*optional*)
- **blocking** (*str*) – Wait for broadcasted transactions to be included in a block and return full transaction (can be “head” or “irreversible”)
- **bundle** (*bool*) – Do not broadcast transactions right away, but allow to bundle operations (*optional*)

Three wallet operation modes are possible:

- **Wallet Database:** Here, the bitshareslibs load the keys from the locally stored wallet SQLite database (see `storage.py`). To use this mode, simply call `BitShares()` without the `keys` parameter
- **Providing Keys:** Here, you can provide the keys for your accounts manually. All you need to do is add the wif keys for the accounts you want to use as a simple array using the `keys` parameter to `BitShares()`.
- **Force keys:** This more is for advanced users and requires that you know what you are doing. Here, the `keys` parameter is a dictionary that overwrite the `active`, `owner`, or `memo` keys for any account. This mode is only used for *foreign* signatures!

If no node is provided, it will connect to the node of <http://uptick.rocks>. It is **highly** recommended that you pick your own node instead. Default settings can be changed with:

```
uptick set node <host>
```

where <host> starts with `ws://` or `wss://`.

The purpose of this class is to simplify interaction with BitShares.

The idea is to have a class that allows to do this:

```
from bitshares import BitShares
bitshares = BitShares()
print(bitshares.info())
```

All that is required is for the user to have added a key with `uptick`

```
uptick addkey
```

and setting a default author:

```
uptick set default_account xeroc
```

This class also deals with edits, votes and reading content.

**account\_whitelist** (*account\_to\_whitelist, lists=None, account=None, \*\*kwargs*)

Account whitelisting.

#### Parameters

- **account\_to\_whitelist** (*str*) – The account we want to add to either the white- or the blacklist
- **lists** (*set*) – (defaults to ('white')). Lists the user should be added to. Either empty set, 'black', 'white' or both.
- **account** (*str*) – (optional) the account to allow access to (defaults to default\_account)

**allow** (*foreign, weight=None, permission='active', account=None, threshold=None, \*\*kwargs*)

Give additional access to an account by some other public key or account.

#### Parameters

- **foreign** (*str*) – The foreign account that will obtain access
- **weight** (*int*) – (optional) The weight to use. If not define, the threshold will be used. If the weight is smaller than the threshold, additional signatures will be required. (defaults to threshold)
- **permission** (*str*) – (optional) The actual permission to modify (defaults to active)
- **account** (*str*) – (optional) the account to allow access to (defaults to default\_account)
- **threshold** (*int*) – The threshold that needs to be reached by signatures to be able to interact

**approvecommittee** (*committees, account=None, \*\*kwargs*)

Approve a committee.

#### Parameters

- **committees** (*list*) – list of committee member name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to default\_account)

**approveproposal** (*proposal\_ids*, *account=None*, *approver=None*, *\*\*kwargs*)

Approve Proposal.

#### Parameters

- **proposal\_id** (*list*) – Ids of the proposals
- **approver** (*str*) – The account or key to use for approval (defaults to *account*)
- **account** (*str*) – (optional) the account to allow access to (defaults to *default\_account*)

**approvewitness** (*witnesses*, *account=None*, *\*\*kwargs*)

Approve a witness.

#### Parameters

- **witnesses** (*list*) – list of Witness name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to *default\_account*)

**approveworker** (*workers*, *account=None*, *\*\*kwargs*)

Approve a worker.

#### Parameters

- **workers** (*list*) – list of worker member name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to *default\_account*)

**asset\_settle** (*amount*, *account=None*, *\*\*kwargs*)

**bid\_collateral** (*additional\_collateral*, *debt\_covered*, *account=None*, *\*\*kwargs*)

**broadcast** (*tx=None*)

Broadcast a transaction to the Blockchain

**Parameters** **tx** (*tx*) – Signed transaction to broadcast

**cancel** (*orderNumbers*, *account=None*, *\*\*kwargs*)

Cancels an order you have placed in a given market. Requires only the “orderNumbers”. An order number takes the form 1.7.xxx.

**Parameters** **orderNumbers** (*str*) – The Order Object ide of the form 1.7.xxxx

**clear** ()

**clear\_cache** ()

Clear Caches

**connect** (*node=*”, *rpcuser=*”, *rpcpassword=*”, *\*\*kwargs*)

Connect to blockchain network (internal use only)

**create\_account** (*account\_name*, *registrar=None*, *referrer='1.2.35641'*, *referrer\_percent=50*, *owner\_key=None*, *active\_key=None*, *memo\_key=None*, *owner\_account=None*, *active\_account=None*, *password=None*, *additional\_owner\_keys=None*, *additional\_active\_keys=None*, *additional\_owner\_accounts=None*, *additional\_active\_accounts=None*, *proxy\_account='proxy-to-self'*, *storekeys=True*, *\*\*kwargs*)

Create new account on BitShares.

The brainkey/password can be used to recover all generated keys (see *bitsharesbase.account* for more details.

By default, this call will use `default_account` to register a new name `account_name` with all keys being derived from a new brain key that will be returned. The corresponding keys will automatically be installed in the wallet.

**Warning:** Don't call this method unless you know what you are doing! Be sure to understand what this method does and where to find the private keys for your account.

---

**Note:** Please note that this imports private keys (if password is present) into the wallet by default. However, it **does not import the owner key** for security reasons. Do NOT expect to be able to recover it from the wallet if you lose your password!

---

#### Parameters

- **account\_name** (*str*) – (required) new account name
- **registrar** (*str*) – which account should pay the registration fee (defaults to `default_account`)
- **owner\_key** (*str*) – Main owner key
- **active\_key** (*str*) – Main active key
- **memo\_key** (*str*) – Main memo\_key
- **password** (*str*) – Alternatively to providing keys, one can provide a password from which the keys will be derived
- **additional\_owner\_keys** (*array*) – Additional owner public keys
- **additional\_active\_keys** (*array*) – Additional active public keys
- **additional\_owner\_accounts** (*array*) – Additional owner account names
- **additional\_active\_accounts** (*array*) – Additional active account names
- **storekeys** (*bool*) – Store new keys in the wallet (default: `True`)

Raises **`AccountExistsException`** – if the account already exists on the blockchain

**create\_asset** (*symbol*, *precision*, *max\_supply*, *description*=", *is\_bitasset*=False, *is\_prediction\_market*=False, *market\_fee\_percent*=0, *max\_market\_fee*=None, *permissions*=None, *flags*=None, *whitelist\_authorities*=None, *blacklist\_authorities*=None, *whitelist\_markets*=None, *blacklist\_markets*=None, *bitasset\_options*=None, *account*=None, *\*\*kwargs*)

Create a new asset.

#### Parameters

- **symbol** (*str*) – Asset symbol
- **precision** (*int*) – Asset precision
- **max\_supply** (*int*) – Asset max supply
- **description** (*str*) – (optional) Asset description
- **is\_bitasset** (*bool*) – (optional) True = bitasset, False = UIA (default: False)
- **is\_prediction\_market** (*bool*) – (optional) True: PD, False = plain smartcoin (default: False)

- **market\_fee\_percent** (*float*) – (optional) Charge market fee (0-100) (default: 0)
- **max\_market\_fee** (*float*) – (optional) Absolute amount of max market fee, value of this option should be a whole number (default: same as max\_supply)
- **permissions** (*dict*) – (optional) Asset permissions
- **flags** (*dict*) – (optional) Enabled asset flags
- **whitelist\_authorities** (*list*) – (optional) List of accounts that serve as whitelist authorities
- **blacklist\_authorities** (*list*) – (optional) List of accounts that serve as blacklist authorities
- **whitelist\_markets** (*list*) – (optional) List of assets to allow trading with
- **blacklist\_markets** (*list*) – (optional) List of assets to prevent trading with
- **bitasset\_options** (*dict*) – (optional) Bitasset settings
- **account** (*str*) – (optional) the issuer account to (defaults to default\_account)

**create\_committee\_member** (*url*=", *account*=None, *\*\*kwargs*)

Create a committee member.

#### Parameters

- **url** (*str*) – URL to read more about the worker
- **account** (*str*) – (optional) the account to allow access to (defaults to default\_account)

**create\_liquidity\_pool** (*asset\_a*, *asset\_b*, *share\_asset*, *taker\_fee\_percent*, *withdrawal\_fee\_percent*, *account*=None, *\*\*kwargs*)

Create a liquidity pool.

#### Parameters

- **asset\_a** (*str*) – First asset in the pool pair.
- **asset\_b** (*str*) – Second asset in the pool pair.
- **share\_asset** (*str*) – The asset which represents shares in the pool.

For asset parameters, these can be either symbols or asset\_id strings. Note that network expects asset\_a to have a lower-numbered asset\_id than asset\_b.

#### Parameters

- **taker\_fee\_percent** (*float*) – The pool's taker fee percentage.
- **withdrawal\_fee\_percent** (*float*) – The pool's withdrawal fee percent.

For percentages, meaningful range is [0.00, 100.00], where 1% is represented as 1.0. Smallest non-zero value recognized by BitShares chain is 0.01 for 0.01%.

**create\_voting\_ticket** (*target\_type*, *amount\_to\_lock*, *account*=None, *\*\*kwargs*)

Create a voting ticket.

#### Parameters

- **target\_type** (*int*, *str*) – Lock period target. Should be a string from operations.ticket\_type\_strings or the index of the intended string.
- **amount\_to\_lock** (*Amount*) – Amount to lock up for the duration selected in target\_type.

**create\_worker** (*name*, *daily\_pay*, *end*, *url*=", *begin*=None, *payment\_type*='vesting',  
*pay\_vesting\_period\_days*=0, *account*=None, *\*\*kwargs*)

Create a worker.

This removes the shares from the supply

### Required

#### Parameters

- **name** (*str*) – Name of the worker
- **daily\_pay** (`bitshares.amount.Amount`) – The amount to be paid daily
- **end** (*datetime*) – Date/time of end of the worker

### Optional

#### Parameters

- **url** (*str*) – URL to read more about the worker
- **begin** (*datetime*) – Date/time of begin of the worker
- **payment\_type** (*string*) – ["burn", "refund", "vesting"] (default: "vesting")
- **pay\_vesting\_period\_days** (*int*) – Days of vesting (default: 0)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**define\_classes** ()

**delete\_liquidity\_pool** (*pool*, *account*=None, *\*\*kwargs*)

Delete a liquidity pool.

**Parameters** **pool** (*str*, `Asset`) – The liquidity pool to delete. Can be the pool id as a string, or can be an Asset, `asset_id`, or symbol of the share asset for the pool.

**deposit\_into\_liquidity\_pool** (*pool*, *amount\_a*, *amount\_b*, *account*=None, *\*\*kwargs*)

Deposit assets into a liquidity pool.

#### Parameters

- **pool** (*str*, `Asset`) – The liquidity pool to use. Can be the pool id as a string, or can be an Asset, `asset_id`, or symbol of the share asset for the pool.
- **amount\_a** (`Amount`) –
- **amount\_b** (`Amount`) –

**disallow** (*foreign*, *permission*='active', *account*=None, *threshold*=None, *\*\*kwargs*)

Remove additional access to an account by some other public key or account.

#### Parameters

- **foreign** (*str*) – The foreign account that will obtain access
- **permission** (*str*) – (optional) The actual permission to modify (defaults to `active`)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)
- **threshold** (*int*) – The threshold that needs to be reached by signatures to be able to interact

**disapprovecommittee** (*committees*, *account*=None, *\*\*kwargs*)

Disapprove a committee.

**Parameters**

- **committees** (*list*) – list of committee name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**disapproveproposal** (*proposal\_ids*, *account=None*, *approver=None*, *\*\*kwargs*)

Disapprove Proposal.

**Parameters**

- **proposal\_ids** (*list*) – Ids of the proposals
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**disapprovewitness** (*witnesses*, *account=None*, *\*\*kwargs*)

Disapprove a witness.

**Parameters**

- **witnesses** (*list*) – list of Witness name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**disapproveworker** (*workers*, *account=None*, *\*\*kwargs*)

Disapprove a worker.

**Parameters**

- **workers** (*list*) – list of worker name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**exchange\_with\_liquidity\_pool** (*pool*, *amount\_to\_sell*, *min\_to\_receive*, *account=None*, *\*\*kwargs*)

Exchange assets against a liquidity pool.

**Parameters**

- **pool** (*str*, *Asset*) – The liquidity pool to use. Can be the pool id as a string, or can be an *Asset*, *asset\_id*, or symbol of the share asset for the pool.
- **amount\_to\_sell** (*Amount*) –
- **min\_to\_receive** (*Amount*) –

**finalizeOp** (*ops*, *account*, *permission*, *\*\*kwargs*)

This method obtains the required private keys if present in the wallet, finalizes the transaction, signs it and broadcasts it

**Parameters**

- **ops** (*operation*) – The operation (or list of operations) to broadcast
- **account** (*operation*) – The account that authorizes the operation
- **permission** (*string*) – The required permission for signing (active, owner, posting)
- **append\_to** (*object*) – This allows to provide an instance of *ProposalsBuilder* (see `new_proposal()`) or *TransactionBuilder* (see `new_tx()`) to specify where to put a specific operation.

... note:: **append\_to** is exposed to every method used in the this class

... note:

If ``ops`` is a list of operation, they all need to be signable by the same key! Thus, you cannot combine ops that require active permission with ops that require posting permission. Neither can you use different accounts for different operations!

... **note::** This uses **txbuffer** as instance of `transactionbuilder.TransactionBuilder`.  
You may want to use your own txbuffer

**fund\_fee\_pool** (*symbol, amount, account=None, \*\*kwargs*)

Fund the fee pool of an asset.

#### Parameters

- **symbol** (*str*) – The symbol to fund the fee pool of
- **amount** (*float*) – The amount to be burned.
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**htlc\_create** (*amount, to, \*args, hash\_type=None, hash\_hex=None, expiration=3600, preimage=None, preimage\_length=0, account=None, \*\*kwargs*)

Create an HTLC contract.

#### Parameters

- **amount** (*Amount*) – Amount to lock
- **to** (*str*) – Recipient
- **expiration** (*int*) – Contract duration in seconds
- **hash\_hex** (*str*) – Hash as string of hex digits
- **preimage** (*str*) – Preimage as ascii string. Note hex digits would be interpreted as ascii text, not as bytes. Not generally recommended to use this option. Options `hash_hex` and `preimage` are mutually exclusive.
- **preimage\_length** (*int*) – If non-zero, htlc contract will require preimage of exact length. Generally OK to leave this as zero. Note if preimage param is provided, this value SHOULD be either zero or match exactly the length of the preimage, else an irredeemable htlc will be created. Optionally, a sentinel value of -1 can be used to compute length automatically from the preimage param.

**htlc\_redeem** (*htlc\_id, preimage, encoding='utf-8', account=None, \*\*kwargs*)

Redeem an htlc contract.

#### Parameters

- **preimage** (*str*) – The preimage that unlocks the htlc
- **encoding** (*str*) – “utf-8”, ..., or “hex”

**info** ()

Returns the global properties

**is\_connected** ()

**newWallet** (*pwd*)



**new\_proposal** (*parent=None, proposer=None, proposal\_expiration=None, proposal\_review=None, \*\*kwargs*)

**new\_tx** (*\*args, \*\*kwargs*)  
Let's obtain a new txbuffer

**Returns int txid** id of the new txbuffer

**new\_wallet** (*pwd*)  
Create a new wallet. This method is basically only calls `wallet.Wallet.create()`.

**Parameters** *pwd* (*str*) – Password to use for the new wallet

**Raises** **exceptions.WalletExists** – if there is already a wallet created

**prefix**  
Contains the prefix of the blockchain

**propbuffer**  
Return the default proposal buffer

**proposal** (*proposer=None, proposal\_expiration=None, proposal\_review=None*)  
Return the default proposal buffer

... note:: If any parameter is set, the default **proposal** parameters will be changed!

**publish\_price\_feed** (*symbol, settlement\_price, cer=None, mssr=110, mcr=200, account=None*)  
Publish a price feed for a market-pegged asset.

**Parameters**

- **symbol** (*str*) – Symbol of the asset to publish feed for
- **settlement\_price** (`bitshares.price.Price`) – Price for settlement
- **cer** (`bitshares.price.Price`) – Core exchange Rate (default `settlement_price + 5%`)
- **mssr** (*float*) – Percentage for max short squeeze ratio (default: 110%)
- **mcr** (*float*) – Percentage for maintenance collateral ratio (default: 200%)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

---

**Note:** The `account` needs to be allowed to produce a price feed for `symbol`. For witness produced feeds this means `account` is a witness account!

---

**reserve** (*amount, account=None, \*\*kwargs*)  
Reserve/Burn an amount of this shares.

This removes the shares from the supply

**Parameters**

- **amount** (`bitshares.amount.Amount`) – The amount to be burned.
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**set\_blocking** (*block=True*)  
This sets a flag that forces the broadcast to block until the transactions made it into a block

**set\_default\_account** (*account*)  
Set the default account to be used

**set\_proxy** (*proxy\_account*, *account=None*, *\*\*kwargs*)

Set a specific proxy for account.

**Parameters**

- **proxy\_account** (*bitshares.account.Account*) – Account to be proxied
- **account** (*str*) – (optional) the account to allow access to (defaults to *default\_account*)

**set\_shared\_instance** ()

This method allows to set the current instance as default

**sign** (*tx=None*, *wifs=[]*)

Sign a provided transaction with the provided key(s)

**Parameters**

- **tx** (*dict*) – The transaction to be signed and returned
- **wifs** (*string*) – One or many wif keys to use for signing a transaction. If not present, the keys will be loaded from the wallet as defined in “missing\_signatures” key of the transactions.

**transfer** (*to*, *amount*, *asset*, *memo=""*, *account=None*, *\*\*kwargs*)

Transfer an asset to another account.

**Parameters**

- **to** (*str*) – Recipient
- **amount** (*float*) – Amount to transfer
- **asset** (*str*) – Asset to transfer
- **memo** (*str*) – (optional) Memo, may begin with # for encrypted messaging
- **account** (*str*) – (optional) the source account for the transfer if not *default\_account*

**tx** ()

Returns the default transaction buffer

**txbuffer**

Returns the currently active tx buffer

**unlock** (*\*args*, *\*\*kwargs*)

Unlock the internal wallet

**unset\_proxy** (*account=None*, *\*\*kwargs*)

Unset the proxy account to start voting yourself.

**update\_cer** (*symbol*, *cer*, *account=None*)

Update the Core Exchange Rate (CER) of an asset.

**Parameters**

- **symbol** (*str*) – Symbol of the asset to publish feed for
- **cer** (*bitshares.price.Price*) – Core exchange Rate
- **account** (*str*) – (optional) the account to allow access to (defaults to *default\_account*)

**update\_memo\_key** (*key*, *account=None*, *\*\*kwargs*)

Update an account’s memo public key.

This method does **not** add any private keys to your wallet but merely changes the memo public key.

#### Parameters

- **key** (*str*) – New memo public key
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**update\_voting\_ticket** (*ticket\_id*, *new\_target\_type*, *amount\_to\_update*, *account=None*, *\*\*kwargs*)

Update a voting ticket.

#### Parameters

- **ticket\_id** (*str*) – Id (e.g. “1.18.xxx”) of the ticket to update.
- **target\_type** (*int*, *str*) – New lock period target. Should be a string from `operations.ticket_type_strings` or the index of the intended string.
- **amount\_to\_update** (*Amount*, *None*) – Amount to move over to the new lock-up target. (Optional - absence implies update whole amount.)

**update\_witness** (*witness\_identifier*, *url=None*, *key=None*, *\*\*kwargs*)

Upgrade a witness account.

#### Parameters

- **witness\_identifier** (*str*) – Identifier for the witness
- **url** (*str*) – New URL for the witness
- **key** (*str*) – Public Key for the signing

**upgrade\_account** (*account=None*, *\*\*kwargs*)

Upgrade an account to Lifetime membership.

**Parameters** **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**vesting\_balance\_withdraw** (*vesting\_id*, *amount=None*, *account=None*, *\*\*kwargs*)

Withdraw vesting balance.

#### Parameters

- **vesting\_id** (*str*) – Id of the vesting object
- **Amount** (*bitshares.amount.Amount*) – to withdraw (“all” if not provided)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**withdraw\_from\_liquidity\_pool** (*pool*, *share\_amount*, *account=None*, *\*\*kwargs*)

Withdraw stake from a liquidity pool.

#### Parameters

- **pool** (*str*, *Asset*) – The liquidity pool to use. Can be the pool id as a string, or can be an *Asset*, *asset\_id*, or symbol of the share asset for the pool.
- **share\_amount** (*Amount*) – Amount of share asset to redeem. Must be a quantity of the pool’s *share\_asset*.

## bitshares.block module

**class** bitshares.block.**Block** (\*args, \*\*kwargs)

Bases: *bitshares.instance.BlockchainInstance*, *bitshares.block.Block*

Read a single block from the chain.

### Parameters

- **block** (*int*) – block number
- **blockchain\_instance** (*bitshares.bitshares.BitShares*) – BitShares instance
- **lazy** (*bool*) – Use lazy loading

Instances of this class are dictionaries that come with additional methods (see below) that allow dealing with a block and it's corresponding functions.

```
from bitshares.block import Block
block = Block(1)
print(block)
```

---

**Note:** This class comes with its own caching function to reduce the load on the API server. Instances of this class can be refreshed with `Account.refresh()`.

---

### bitshares

Alias for the specific blockchain.

### blockchain

### blockchain\_instance\_class

alias of *bitshares.instance.BlockchainInstance*

**classmethod** **cache\_object** (*data*, *key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

### chain

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod** **clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier** = None

**incached**(*id*)

Is an element cached?

**classmethod inject**(*cls*)

**items**()

This overwrites items() so that refresh() is called if the object is not already fetched

**keys**() → a set-like object providing a view on D's keys

**static objectid\_valid**(*i*)

Test if a string looks like a regular object id of the form::

```
xxxx.yyyyy.zzzz
```

with those being numbers.

**perform\_id\_tests** = True

**pop**(*k*, [*d*]) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise KeyError is raised

**popitem**() → (*k*, *v*), remove and return some (key, value) pair as a

2-tuple; but raise KeyError if D is empty.

**refresh**()

Even though blocks never change, you freshly obtain its contents from an API with this method

**static set\_cache\_store**(*klass*, \**args*, \*\**kwargs*)

**classmethod set\_shared\_blockchain\_instance**(*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config**(*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance**()

This method allows to set the current instance as default

**setdefault**()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance**()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store**(*data*, *key*='id')

Cache the list

**Parameters** *data* (*list*) – List of objects to cache

**test\_valid\_objectid**(*i*)

Alias for `objectid_valid`

**testid** (*id*)

In contrast to validity, this method tests if the objectid matches the type\_id provided in self.type\_id or self.type\_ids

**time** ()

Return a datetime instance for the timestamp of this block

**type\_id** = 'n/a'

**type\_ids** = []

**update** ([*E*], **\*\*F**) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitshares.block.**BlockHeader** (\*args, **\*\*kwargs**)

Bases: *bitshares.instance.BlockchainInstance*, *bitshares.block.BlockHeader*

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of *bitshares.instance.BlockchainInstance*

**classmethod** **cache\_object** (*data*, *key=None*)

This classmethod allows to feed an object into the cache is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod** **clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier** = None

**incached** (*id*)

Is an element cached?

**classmethod** **inject** (*cls*)

**items** ()

This overwrites items() so that refresh() is called if the object is not already fetched

**keys()** → a set-like object providing a view on D's keys

**static objectid\_valid(i)**

Test if a string looks like a regular object id of the form::

```
xxxx.yyyyy.zzzz
```

with those being numbers.

**perform\_id\_tests = True**

**pop(k[, d])** → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

**popitem()** → (k, v), remove and return some (key, value) pair as a

2-tuple; but raise KeyError if D is empty.

**refresh()**

Even though blocks never change, you freshly obtain its contents from an API with this method

**static set\_cache\_store(klass, \*args, \*\*kwargs)**

**classmethod set\_shared\_blockchain\_instance(instance)**

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters instance(chaininstance)** – Chain instance

**classmethod set\_shared\_config(config)**

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id = 1**

**store(data, key='id')**

Cache the list

**Parameters data(list)** – List of objects to cache

**test\_valid\_objectid(i)**

Alias for `objectid_valid`

**testid(id)**

In contrast to validity, this method tests if the objectid matches the `type_id` provided in `self.type_id` or `self.type_ids`

**time()**

Return a datetime instance for the timestamp of this block

**type\_id = 'n/a'**

**type\_ids = []**

**update** (*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

## bitshares.blockchain module

**class** `bitshares.blockchain.Blockchain` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.blockchain.Blockchain`

This class allows to access the blockchain and read data from it.

### Parameters

- **blockchain\_instance** (`bitshares.bitshares.BitShares`) – BitShares instance
- **mode** (*str*) – (default) Irreversible block (`irreversible`) or actual head block (`head`)
- **max\_block\_wait\_repetition** (*int*) – (default) 3 maximum wait time for next block  
is `max_block_wait_repetition * block_interval`

This class let's you deal with blockchain related data and methods.

**awaitTxConfirmation** (*transaction*, *limit=10*)

Returns the transaction as seen by the blockchain after being included into a block

---

**Note:** If you want instant confirmation, you need to instantiate class: `blockchain.Blockchain` with `mode="head"`, otherwise, the call will wait until confirmed in an irreversible block.

---

---

**Note:** This method returns once the blockchain has included a transaction with the **same signature**. Even though the signature is not usually used to identify a transaction, it still cannot be forfeited and is derived from the transaction content and thus identifies a transaction uniquely.

---

## bitshares

Alias for the specific blockchain.

**block\_time** (*block\_num*)

Returns a datetime of the block with the given block number.

Parameters **block\_num** (*int*) – Block number

**block\_timestamp** (*block\_num*)

Returns the timestamp of the block with the given block number.

Parameters **block\_num** (*int*) – Block number

## blockchain

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**blocks** (*start=None*, *stop=None*)

Yields blocks starting from *start*.

### Parameters

- **start** (*int*) – Starting block



- **stop** (*int*) – Stop at this block
- **mode** (*str*) – We here have the choice between “head” (the last block) and “irreversible” (the block that is confirmed by 2/3 of all block producers and is thus irreversible)

**chain**

Short form for blockchain (for the lazy)

**chainParameters** ()

The blockchain parameters, such as fees, and committee-controlled parameters are returned here

**config** ()

Returns object 2.0.0

**define\_classes** ()

Needs to define instance variables that provide classes

**get\_all\_accounts** (*start=*”, *stop=*”, *steps=1000.0*, *\*\*kwargs*)

Yields account names between start and stop.

**Parameters**

- **start** (*str*) – Start at this account name
- **stop** (*str*) – Stop at this account name
- **steps** (*int*) – Obtain *steps* ret with a single call from RPC

**get\_block\_interval** ()

This call returns the block interval

**get\_chain\_properties** ()

Return chain properties

**get\_current\_block** ()

This call returns the current block

---

**Note:** The block number returned depends on the *mode* used when instanciating from this class.

---

**get\_current\_block\_num** ()

This call returns the current block

---

**Note:** The block number returned depends on the *mode* used when instanciating from this class.

---

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**get\_network** ()

Identify the network

**Returns** Network parameters

**Return type** dict

**info** ()

This call returns the *dynamic global properties*

**classmethod inject** (*cls*)**is\_irreversible\_mode** ()

**ops** (*start=None, stop=None, \*\*kwargs*)

Yields all operations (excluding virtual operations) starting from *start*.

**Parameters**

- **start** (*int*) – Starting block
- **stop** (*int*) – Stop at this block
- **mode** (*str*) – We here have the choice between “head” (the last block) and “irreversible” (the block that is confirmed by 2/3 of all block producers and is thus irreversible)
- **only\_virtual\_ops** (*bool*) – Only yield virtual operations

This call returns a list that only carries one operation and its type!

**participation\_rate**

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**stream** (*opNames=[], \*args, \*\*kwargs*)

Yield specific operations (e.g. comments) only

**Parameters**

- **opNames** (*array*) – List of operations to filter for
- **start** (*int*) – Start at this block
- **stop** (*int*) – Stop at this block
- **mode** (*str*) – We here have the choice between \* “head”: the last block \* “irreversible”: the block that is confirmed by 2/3 of all block producers and is thus irreversible!

The dict output is formatted such that `type` carries the operation type, `timestamp` and `block_num` are taken from the block the operation was stored in and the other key depend on the actualy operation.

**update\_chain\_parameters** ()

**wait\_for\_and\_get\_block** (*block\_number, blocks\_waiting\_for=None*)

Get the desired block from the chain, if the current head block is smaller (for both head and irreversible) then we wait, but a maximum of `blocks_waiting_for * max_block_wait_repetition` time before failure.

**Parameters**

- **block\_number** (*int*) – desired block number
- **blocks\_waiting\_for** (*int*) – (default) difference between `block_number` and current head how many blocks we are willing to wait, positive int

**bitshares.blockchainobject module**

**class** bitshares.blockchainobject.**BlockchainObject** (\*args, \*\*kwargs)  
 Bases: *bitshares.instance.BlockchainInstance*, *bitshares.blockchainobject.BlockchainObject*

**bitshares**  
 Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**  
 alias of *bitshares.instance.BlockchainInstance*

**classmethod** **cache\_object** (data, key=None)  
 This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**  
 Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod** **clear\_cache** ()  
 Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()  
 Needs to define instance variables that provide classes

**fromkeys** ()  
 Create a new dictionary with keys from iterable and values set to value.

**get** ()  
 Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()  
 Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (id)  
 Get an element from the cache explicitly

**identifier** = None

**incached** (id)  
 Is an element cached?

**classmethod** **inject** (cls)

**items** ()  
 This overwrites items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static** **objectid\_valid** (i)  
 Test if a string looks like a regular object id of the form::

```
xxxx.yyzz.yyzz.zzzz
```

with those being numbers.

**perform\_id\_tests** = True

**pop** (*k*, *d*) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if *D* is empty.

**static set\_cache\_store** (*klass*, *\*args*, *\*\*kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (*data*, *key*='id')

Cache the list

**Parameters** *data* (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)

Alias for `objectid_valid`

**testid** (*id*)

In contrast to validity, this method tests if the objectid matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id** = None

**type\_ids** = []

**update** ([*E*], *\*\*F*) → None. Update *D* from dict/iterable *E* and *F*.

If *E* is present and has a `.keys()` method, then does: for *k* in *E*: *D*[*k*] = *E*[*k*] If *E* is present and lacks a `.keys()` method, then does: for *k*, *v* in *E*: *D*[*k*] = *v* In either case, this is followed by: for *k* in *F*: *D*[*k*] = *F*[*k*]

**values** () → an object providing a view on *D*'s values

**class** `bitshares.blockchainobject.Object` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.blockchainobject.Object`

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**classmethod** `cache_object (data, key=None)`

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod** `clear_cache ()`

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier** = None

**incached** (*id*)

Is an element cached?

**classmethod** `inject (cls)`

**items** ()

This overwrites items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static** `objectid_valid (i)`

Test if a string looks like a regular object id of the form::

```
xxxx.yyzz.zzzz
```

with those being numbers.

**perform\_id\_tests** = False

**pop** (*k*, *d*) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise *KeyError* is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a

2-tuple; but raise *KeyError* if D is empty.

**refresh** ()

This is the refresh method that overloads the prototype in *BlockchainObject*.

**static** `set_cache_store (klass, *args, **kwargs)`

**classmethod** `set_shared_blockchain_instance (instance)`

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod `set_shared_config`** (*config*)  
This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**`set_shared_instance`** ()  
This method allows to set the current instance as default

**`setdefault`** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**`shared_blockchain_instance`** ()  
This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**`space_id`** = 1

**`store`** (*data*, *key*='id')  
Cache the list  
  
**Parameters** *data* (*list*) – List of objects to cache

**`test_valid_objectid`** (*i*)  
Alias for `objectid_valid`

**`testid`** (*id*)  
In contrast to validity, this method tests if the `objectid` matches the `type_id` provided in `self.type_id` or `self.type_ids`

**`type_id`** = None

**`type_ids`** = []

**`update`** ([*E*], \*\**F*) → None. Update D from dict/iterable E and F.  
If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**`values`** () → an object providing a view on D's values

## bitshares.committee module

**class** `bitshares.committee.Committee` (\*args, \*\*kwargs)  
Bases: `bitshares.instance.BlockchainInstance`, `bitshares.committee.Committee`  
Read data about a Committee Member in the chain.

**Parameters**

- **`member`** (*str*) – Name of the Committee Member
- **`blockchain_instance`** (*bitshares*) – BitShares() instance to use when accesing a RPC
- **`lazy`** (*bool*) – Use lazy loading

**`account`**  
**`account_id`**  
**`bitshares`**  
Alias for the specific blockchain.  
**`blockchain`**

**blockchain\_instance\_class**

alias of *bitshares.instance.BlockchainInstance*

**classmethod cache\_object** (*data, key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier = None**

**incached** (*id*)

Is an element cached?

**classmethod inject** (*cls*)

**items** ()

This overwrites items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static objectid\_valid** (*i*)

Test if a string looks like a regular object id of the form::

```
xxxx.yyzz.zzzz
```

with those being numbers.

**perform\_id\_tests = True**

**pop** (*k[, d]*) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

**popitem** () → (k, v), remove and return some (key, value) pair as a

2-tuple; but raise KeyError if D is empty.

**refresh** ()

**static set\_cache\_store** (*klass, \*args, \*\*kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters instance** (*chaininstance*) – Chain instance

**classmethod** `set_shared_config (config)`

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (data, key='id')

Cache the list

**Parameters** `data (list)` – List of objects to cache

**test\_valid\_objectid** (i)

Alias for `objectid_valid`

**testid** (id)

In contrast to validity, this method tests if the objectid matches the type\_id provided in `self.type_id` or `self.type_ids`

**type\_id** = None

**type\_ids** = []

**update** ([E], \*\*F) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

## bitshares.dex module

**class** `bitshares.dex.Dex (*args, **kwargs)`

Bases: `bitshares.instance.BlockchainInstance`

This class simplifies interactions with the decentralized exchange.

**Parameters** `blockchain_instance (bitshares.bitshares.BitShares)` – BitShares instance

---

**Note:** The methods of this class only deal with a single asset (at most). If you are looking to deal with orders for trading, please use `bitshares.market.Market`.

---

**adjust\_collateral\_ratio** (symbol, new\_collateral\_ratio, account=None, target\_collateral\_ratio=None)

Adjust the collateral ratio of a debt position.

**Parameters**

- **symbol** (str) – Symbol to adjust collateral for



- **new\_collateral\_ratio** (*float*) – desired collateral ratio
- **target\_collateral\_ratio** (*float*) – Tag the call order so that in case of margin call, only enough debt is covered to get back to this ratio

**Raises**

- **ValueError** – if symbol is not a bitasset
- **ValueError** – if collateral ratio is smaller than maintenance collateral ratio
- **ValueError** – if required amounts of collateral are not available

**adjust\_debt** (*delta*, *new\_collateral\_ratio=None*, *account=None*, *target\_collateral\_ratio=None*)

Adjust the amount of debt for an asset.

**Parameters**

- **delta** (*Amount*) – Delta amount of the debt (-10 means reduce debt by 10, +10 means borrow another 10)
- **new\_collateral\_ratio** (*float*) – collateral ratio to maintain (optional, by default tries to maintain old ratio)
- **target\_collateral\_ratio** (*float*) – Tag the call order so that in case of margin call, only enough debt is covered to get back to this ratio

**Raises**

- **ValueError** – if symbol is not a bitasset
- **ValueError** – if collateral ratio is smaller than maintenance collateral ratio
- **ValueError** – if required amounts of collateral are not available

**bitshares**

Alias for the specific blockchain.

**blockchain**

**borrow** (*amount*, *collateral\_ratio=None*, *account=None*, *target\_collateral\_ratio=None*)

Borrow bitassets/smartcoins from the network by putting up collateral in a CFD at a given collateral ratio.

**Parameters**

- **amount** (*Amount*) – Amount to borrow (denoted in 'asset')
- **collateral\_ratio** (*float*) – Collateral ratio to borrow at
- **target\_collateral\_ratio** (*float*) – Tag the call order so that in case of margin call, only enough debt is covered to get back to this ratio

**Raises**

- **ValueError** – if symbol is not a bitasset
- **ValueError** – if collateral ratio is smaller than maintenance collateral ratio
- **ValueError** – if required amounts of collateral are not available

**chain**

Short form for blockchain (for the lazy)

**close\_debt\_position** (*symbol*, *account=None*)

Close a debt position and reclaim the collateral.

**Parameters** **symbol** (*str*) – Symbol to close debt position for

**Raises ValueError** – if symbol has no open call position

**define\_classes()**

Needs to define instance variables that provide classes

**get\_instance\_class()**

Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject(cls)**

**list\_debt\_positions(account=None)**

List Call Positions (borrowed assets and amounts)

**Returns** Struct of assets with amounts and call price

**Return type** dict

**Example:**

**returnFees()**

Returns a dictionary of all fees that apply through the network.

Example output:

```
{'proposal_create': {'fee': 400000.0},
'asset_publish_feed': {'fee': 1000.0}, 'account_create':
{'basic_fee': 950000.0, 'price_per_kbyte': 20000.0,
'premium_fee': 40000000.0}, 'custom': {'fee': 20000.0},
'asset_fund_fee_pool': {'fee': 20000.0},
'override_transfer': {'fee': 400000.0}, 'fill_order':
{}, 'asset_update': {'price_per_kbyte': 20000.0, 'fee':
200000.0}, 'asset_update_feed_producers': {'fee':
10000000.0}, 'assert': {'fee': 20000.0},
'committee_member_create': {'fee': 100000000.0}}
```

**classmethod set\_shared\_blockchain\_instance(instance)**

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters instance(chaininstance)** – Chain instance

**classmethod set\_shared\_config(config)**

This allows to set a config that will be used when calling *shared\_blockchain\_instance* and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**shared\_blockchain\_instance()**

This method will initialize *SharedInstance.instance* and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

## bitshares.exceptions module

**exception bitshares.exceptions.AccountExistsException**

Bases: *Exception*

The requested account already exists.

**args**

**with\_traceback()**

*Exception.with\_traceback(tb)* – set *self.\_\_traceback\_\_* to *tb* and return *self*.

**exception** `bitshares.exceptions.HtlcDoesNotExistException`

Bases: `Exception`

HTLC object does not exist.

**args**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception** `bitshares.exceptions.ObjectNotInProposalBuffer`

Bases: `Exception`

Object was not found in proposal.

**args**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception** `bitshares.exceptions.RPCConnectionRequired`

Bases: `Exception`

An RPC connection is required.

**args**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

## bitshares.genesisbalance module

**class** `bitshares.genesisbalance.GenesisBalance(*args, **kwargs)`

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.genesisbalance.GenesisBalance`

Read data about a Genesis Balances from the chain.

### Parameters

- **identifier** (*str*) – identifier of the balance
- **blockchain\_instance** (*bitshares*) – bitshares() instance to use when accessing a RPC

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**classmethod** **cache\_object** (*data*, *key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**claim** (*account=None*, *\*\*kwargs*)

Claim a balance from the genesis block

### Parameters

- **balance\_id** (*str*) – The identifier that identifies the balance to claim (1.15.x)
- **account** (*str*) – (optional) the account that owns the bet (defaults to `default_account`)

**clear** () → None. Remove all items from D.

**classmethod clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier** = None

**incached** (*id*)

Is an element cached?

**classmethod inject** (*cls*)

**items** ()

This overwrites items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static objectid\_valid** (*i*)

Test if a string looks like a regular object id of the form::

`xxxx.yyyyy.zzzz`

with those being numbers.

**perform\_id\_tests** = True

**pop** (*k*[, *d*]) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a

2-tuple; but raise `KeyError` if D is empty.

**refresh** ()

**static set\_cache\_store** (*klass*, *\*args*, *\*\*kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id = 1**

**store** (*data*, *key*='id')

Cache the list

**Parameters** *data* (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)

Alias for `objectid_valid`

**testid** (*id*)

In contrast to validity, this method tests if the objectid matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id = 15**

**type\_ids = []**

**update** (*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** `bitshares.genesisbalance.GenesisBalances` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.genesisbalance.GenesisBalances`

List genesis balances that can be claimed from the keys in the wallet.

**append** ()

Append object to the end of the list.

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**chain**

Short form for blockchain (for the lazy)

**clear** ()

Remove all items from list.

**copy** ()

Return a shallow copy of the list.

**count** ()

Return number of occurrences of value.

**define\_classes()**

Needs to define instance variables that provide classes

**extend()**

Extend list by appending elements from the iterable.

**get\_instance\_class()**

Should return the Chain instance class, e.g. *bitshares.BitShares*

**index()**

Return first index of value.

Raises *ValueError* if the value is not present.

**classmethod inject(cls)**

**insert()**

Insert object before index.

**pop()**

Remove and return item at index (default last).

Raises *IndexError* if list is empty or index is out of range.

**remove()**

Remove first occurrence of value.

Raises *ValueError* if the value is not present.

**reverse()**

Reverse *IN PLACE*.

**classmethod set\_shared\_blockchain\_instance(instance)**

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters** *instance(chaininstance)* – Chain instance

**classmethod set\_shared\_config(config)**

This allows to set a config that will be used when calling *shared\_blockchain\_instance* and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**shared\_blockchain\_instance()**

This method will initialize *SharedInstance.instance* and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**sort()**

Stable sort *IN PLACE*.

## bitshares.htlc module

**class** *bitshares.htlc.Htlc*(\*args, \*\*kwargs)

Bases: *bitshares.blockchainobject.BlockchainObject*

Read data about an HTLC contract on the chain.

**Parameters**

- **id(str)** – id of the HTLC

- **blockchain\_instance** (*bitshares*) – BitShares() instance to use when accessing a RPC

**bitshares**

Alias for the specific blockchain.

**blockchain****blockchain\_instance\_class**

alias of *bitshares.instance.BlockchainInstance*

**classmethod cache\_object** (*data, key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.**classmethod clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier = None****incached** (*id*)

Is an element cached?

**classmethod inject** (*cls*)**items** ()

This overwrites items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys**static objectid\_valid** (*i*)

Test if a string looks like a regular object id of the form::

```
xxxx.yyzz.zzzz
```

with those being numbers.

**perform\_id\_tests = True****pop** (*k[, d]*) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

**popitem** () → (k, v), remove and return some (key, value) pair as a

2-tuple; but raise KeyError if D is empty.

**refresh()**

**static set\_cache\_store** (*klass*, \**args*, \*\**kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (*data*, *key*='id')

Cache the list

**Parameters** *data* (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)

Alias for `objectid_valid`

**testid** (*id*)

In contrast to validity, this method tests if the `objectid` matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id** = 16

**type\_ids** = []

**update** ([*E*], \*\**F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

## bitshares.instance module

**class** `bitshares.instance.BlockchainInstance` (\**args*, \*\**kwargs*)

Bases: `graphenecommon.instance.AbstractBlockchainInstanceProvider`

This is a class that allows compatibility with previous naming conventions.

**bitshares**

Alias for the specific blockchain.

**blockchain**

**chain**

Short form for blockchain (for the lazy)



**define\_classes()**

Needs to define instance variables that provide classes

**get\_instance\_class()**

Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject(cls)**

**classmethod set\_shared\_blockchain\_instance(instance)**

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config(config)**

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**class** `bitshares.instance.SharedInstance`

Bases: `object`

This class merely offers a singleton for the Blockchain Instance.

**config** = {}

**instance** = None

`bitshares.instance.set_shared_bitshares_instance(instance)`

`bitshares.instance.set_shared_blockchain_instance(instance)`

`bitshares.instance.set_shared_config(config)`

`bitshares.instance.shared_bitshares_instance()`

`bitshares.instance.shared_blockchain_instance()`

## bitshares.market module

**class** `bitshares.market.Market(*args, **kwargs)`

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.market.Market`

This class allows to easily access Markets on the blockchain for trading, etc.

### Parameters

- **blockchain\_instance** (`bitshares.bitshares.BitShares`) – BitShares instance
- **base** (`bitshares.asset.Asset`) – Base asset
- **quote** (`bitshares.asset.Asset`) – Quote asset

**Returns** Blockchain Market

**Return type** dictionary with overloaded methods

Instances of this class are dictionaries that come with additional methods (see below) that allow dealing with a market and it's corresponding functions.

This class tries to identify **two** assets as provided in the parameters in one of the following forms:

- `base` and `quote` are valid assets (according to `bitshares.asset.Asset`)
- `base:quote` separated with `:`
- `base/quote` separated with `/`
- `base-quote` separated with `-`

---

**Note:** Throughout this library, the `quote` symbol will be presented first (e.g. `USD:BTS` with `USD` being the quote), while the `base` only refers to a secondary asset for a trade. This means, if you call `bitshares.market.Market.sell()` or `bitshares.market.Market.buy()`, you will sell/buy **only quote** and obtain/pay **only base**.

---

**accountopenorders** (*account=None*)

Returns open Orders.

**Parameters** **account** (`bitshares.account.Account`) – Account name or instance of Account to show orders for in this market

**accounttrades** (*account=None, limit=25*)

Returns your trade history for a given market, specified by the “currencyPair” parameter. You may also specify “all” to get the orderbooks of all markets.

**Parameters**

- **currencyPair** (*str*) – Return results for a particular market only (default: “all”)
- **limit** (*int*) – Limit the amount of orders (default: 25)

Output Parameters:

- *type*: sell or buy
- *rate*: price for *quote* denoted in *base* per *quote*
- *amount*: amount of quote
- *total*: amount of base at asked price (amount/price)

---

**Note:** This call goes through the trade history and searches for your account, if there are no orders within `limit` trades, this call will return an empty array.

---

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**buy** (*price, amount, expiration=None, killfill=False, account=None, returnOrderId=False, \*\*kwargs*)

Places a buy order in a given market.

**Parameters**

- **price** (*float*) – price denoted in base/quote

- **amount** (*number*) – Amount of quote to buy
- **expiration** (*number*) – (optional) expiration time of the order in seconds (defaults to 7 days)
- **killfill** (*bool*) – flag that indicates if the order shall be killed if it is not filled (defaults to False)
- **account** (*string*) – Account name that executes that order
- **returnOrderId** (*string*) – If set to “head” or “irreversible” the call will wait for the tx to appear in the head/irreversible block and add the key “orderid” to the tx output

Prices/Rates are denoted in ‘base’, i.e. the USD\_BTS market is priced in BTS per USD.

**Example:** in the USD\_BTS market, a price of 300 means a USD is worth 300 BTS

**Note:** All prices returned are in the **reversed** orientation as the market. I.e. in the BTC/BTS market, prices are BTS per BTC. That way you can multiply prices with *1.05* to get a +5%.

**Warning:** Since buy orders are placed as limit-sell orders for the base asset, you may end up obtaining more of the buy asset than you placed the order for. Example:

- You place an order to buy 10 USD for 100 BTS/USD
- This means that you actually place a sell order for 1000 BTS in order to obtain **at least** 10 USD
- If an order on the market exists that sells USD for cheaper, you will end up with more than 10 USD

**cancel** (*orderNumber*, *account=None*, *\*\*kwargs*)

Cancels an order you have placed in a given market. Requires only the “orderNumber”. An order number takes the form *1.7.xxx*.

**Parameters** **orderNumber** (*str*) – The Order Object id of the form *1.7.xxxx*

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**copy** () → a shallow copy of D

**core\_base\_market** ()

This returns an instance of the market that has the core market of the base asset.

It means that base needs to be a market pegged asset and returns a market to its collateral asset.

**core\_quote\_market** ()

This returns an instance of the market that has the core market of the quote asset.

It means that quote needs to be a market pegged asset and returns a market to its collateral asset.

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class()**

Should return the Chain instance class, e.g. *bitshares.BitShares*

**get\_limit\_orders** (*limit*=25)

Returns the list of limit orders for a given market.

**Parameters** *limit* (*int*) – Limit the amount of orders (default: 25)

Sample output:

```
[0.003679 USD/BTS (1.9103 USD|519.29602 BTS),
0.003676 USD/BTS (299.9997 USD|81606.16394 BTS),
0.003665 USD/BTS (288.4618 USD|78706.21881 BTS),
0.003665 USD/BTS (3.5285 USD|962.74409 BTS),
0.003665 USD/BTS (72.5474 USD|19794.41299 BTS),
[0.003738 USD/BTS (36.4715 USD|9756.17339 BTS),
0.003738 USD/BTS (18.6915 USD|5000.00000 BTS),
0.003742 USD/BTS (182.6881 USD|48820.22081 BTS),
0.003772 USD/BTS (4.5200 USD|1198.14798 BTS),
0.003799 USD/BTS (148.4975 USD|39086.59741 BTS)]
```

---

**Note:** Each bid is an instance of class:*bitshares.price.Order* and thus carries the keys *base*, *quote* and *price*. From those you can obtain the actual amounts for sale

---

**get\_string** (*separator*=':')

Return a formatted string that identifies the market, e.g. USD:BTS

**Parameters** *separator* (*str*) – The separator of the assets (defaults to :)

**classmethod inject** (*cls*)

**items** () → a set-like object providing a view on D's items

**keys** () → a set-like object providing a view on D's keys

**orderbook** (*limit*=25)

Returns the order book for a given market. You may also specify “all” to get the orderbooks of all markets.

**Parameters** *limit* (*int*) – Limit the amount of orders (default: 25)

Sample output:

```
{'bids': [0.003679 USD/BTS (1.9103 USD|519.29602 BTS),
0.003676 USD/BTS (299.9997 USD|81606.16394 BTS),
0.003665 USD/BTS (288.4618 USD|78706.21881 BTS),
0.003665 USD/BTS (3.5285 USD|962.74409 BTS),
0.003665 USD/BTS (72.5474 USD|19794.41299 BTS)],
'asks': [0.003738 USD/BTS (36.4715 USD|9756.17339 BTS),
0.003738 USD/BTS (18.6915 USD|5000.00000 BTS),
0.003742 USD/BTS (182.6881 USD|48820.22081 BTS),
0.003772 USD/BTS (4.5200 USD|1198.14798 BTS),
0.003799 USD/BTS (148.4975 USD|39086.59741 BTS)]}
```

---

**Note:** Each bid is an instance of class:*bitshares.price.Order* and thus carries the keys *base*, *quote* and *price*. From those you can obtain the actual amounts for sale

---

---

**Note:** This method does order consolidation and hides some details of individual orders!

---

**pop** (*k*, *d*) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if *D* is empty.

**sell** (*price*, *amount*, *expiration=None*, *killfill=False*, *account=None*, *returnOrderId=False*, *\*\*kwargs*)

Places a sell order in a given market.

#### Parameters

- **price** (*float*) – price denoted in base/quote
- **amount** (*number*) – Amount of quote to sell
- **expiration** (*number*) – (optional) expiration time of the order in seconds (defaults to 7 days)
- **killfill** (*bool*) – flag that indicates if the order shall be killed if it is not filled (defaults to `False`)
- **account** (*string*) – Account name that executes that order
- **returnOrderId** (*string*) – If set to “head” or “irreversible” the call will wait for the tx to appear in the head/irreversible block and add the key “orderid” to the tx output

Prices/Rates are denoted in ‘base’, i.e. the USD\_BTS market is priced in BTS per USD.

**Example:** in the USD\_BTS market, a price of 300 means a USD is worth 300 BTS

---

**Note:** All prices returned are in the **reversed** orientation as the market. I.e. in the BTC/BTS market, prices are BTS per BTC. That way you can multiply prices with *1.05* to get a +5%.

---

**classmethod** **set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** **instance** (*chaininstance*) – Chain instance

**classmethod** **set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**ticker** ()

Returns the ticker for all markets.

Output Parameters:

- **last**: Price of the order last filled
- **lowestAsk**: Price of the lowest ask
- **highestBid**: Price of the highest bid
- **baseVolume**: Volume of the base asset
- **quoteVolume**: Volume of the quote asset
- **percentChange**: 24h change percentage (in %)
- **settlement\_price**: Settlement Price for borrow/settlement
- **core\_exchange\_rate**: Core exchange rate for payment of fee in non-BTS asset
- **price24h**: the price 24h ago

Sample Output:

```
{
  {
    "quoteVolume": 48328.73333,
    "quoteSettlement_price": 332.3344827586207,
    "lowestAsk": 340.0,
    "baseVolume": 144.1862,
    "percentChange": -1.9607843231354893,
    "highestBid": 334.20000000000005,
    "latest": 333.3333330133934,
  }
}
```

**trades** (*limit=25, start=None, stop=None*)

Returns your trade history for a given market.

#### Parameters

- **limit** (*int*) – Limit the amount of orders (default: 25)
- **start** (*datetime*) – start time
- **stop** (*datetime*) – stop time

**update** (*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**volume24h** ()

Returns the 24-hour volume for all markets, plus totals for primary currencies.

Sample output:

```
{
  "BTS": 361666.63617,
  "USD": 1087.0
}
```

## bitshares.memo module

**class** bitshares.memo.**Memo** (*\*args, \*\*kwargs*)

Bases: *bitshares.instance.BlockchainInstance, bitshares.memo.Memo*

Deals with Memos that are attached to a transfer.

#### Parameters

- **from\_account** (`bitshares.account.Account`) – Account that has sent the memo
- **to\_account** (`bitshares.account.Account`) – Account that has received the memo
- **blockchain\_instance** (`bitshares.bitshares.BitShares`) – BitShares instance

A memo is encrypted with a shared secret derived from a private key of the sender and a public key of the receiver. Due to the underlying mathematics, the same shared secret can be derived by the private key of the receiver and the public key of the sender. The encrypted message is perturbed by a nonce that is part of the transmitted message.

```
from bitshares.memo import Memo
m = Memo("bitshareseu", "wallet.xeroc")
m.unlock_wallet("secret")
enc = (m.encrypt("foobar"))
print(enc)
>> {'nonce': '17329630356955254641', 'message': '8563e2bb2976e0217806d642901a2855
↪'}
print(m.decrypt(enc))
>> foobar
```

To decrypt a memo, simply use

```
from bitshares.memo import Memo
m = Memo()
m.blockchain.wallet.unlock("secret")
print(memo.decrypt(op_data["memo"]))
```

if `op_data` being the payload of a transfer operation.

#### bitshares

Alias for the specific blockchain.

#### blockchain

#### blockchain\_instance\_class

alias of `bitshares.instance.BlockchainInstance`

#### chain

Short form for blockchain (for the lazy)

#### decrypt (message)

Decrypt a message

**Parameters** `message` (*dict*) – encrypted memo message

**Returns** decrypted message

**Return type** str

#### define\_classes ()

Needs to define instance variables that provide classes

#### encrypt (message)

Encrypt a memo

**Parameters** `message` (*str*) – clear text memo message

**Returns** encrypted message

**Return type** str

```
get_instance_class()
```

Should return the Chain instance class, e.g. *bitshares.BitShares*

```
classmethod inject (cls)
```

```
classmethod set_shared_blockchain_instance(instance)
```

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** **instance** (*chaininstance*) – Chain instance

```
classmethod set_shared_config(config)
```

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

```
set_shared_instance()
```

This method allows to set the current instance as default

```
shared_blockchain_instance()
```

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

```
unlock_wallet (*args, **kwargs)
```

Unlock the library internal wallet

## bitshares.message module

```
class bitshares.message.Message(*args, **kwargs)
```

Bases: *bitshares.instance.BlockchainInstance*, *bitshares.message.Message*

```
MESSAGE_SPLIT = ('-----BEGIN BITSHARES SIGNED MESSAGE-----', '-----BEGIN META-----', '.')
```

```
SIGNED_MESSAGE_ENCAPSULATED = '\n{MESSAGE_SPLIT[0]}\n{message}\n{MESSAGE_SPLIT[1]}\nac'
```

bitshares

Alias for the specific blockchain.

blockchain

```
blockchain_instance_class
```

alias of `bitshares.instance.BlockchainInstance`

chain

## Short form for blockchain (for the lazy)

```
define classes()
```

Needs to define instance variables that provide classes

```
get_instance_class()
```

Should return the Chain instance class, e.g. *bitshares.BitShares*

```
classmethod inject (cls)
```

```
classmethod set_shared_blockchain_instance(instance)
```

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** `instance` (*chaininstance*) – Chain instance



**classmethod** `set_shared_config (config)`

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**sign** (\*args, \*\*kwargs)

Sign a message with an account's memo key

**Parameters** `account` (*str*) – (optional) the account that owns the bet (defaults to `default_account`)

**Raises** `ValueError` – If not account for signing is provided

**Returns** the signed message encapsulated in a known format

**supported\_formats** = (<class 'graphenecommon.message.MessageV1'>, <class 'graphenecommon

**valid\_exceptions** = (<class 'graphenecommon.exceptions.AccountDoesNotExistsException'>,

**verify** (\*\*kwargs)

Verify a message with an account's memo key

**Parameters** `account` (*str*) – (optional) the account that owns the bet (defaults to `default_account`)

**Returns** True if the message is verified successfully

:raises `InvalidMessageSignature` if the signature is not ok

## bitshares.notify module

**class** `bitshares.notify.Notify` (*accounts=None, markets=None, objects=None, on\_tx=None, on\_object=None, on\_block=None, on\_account=None, on\_market=None, keep\_alive=25, \*\*kwargs*)

Bases: `events.events.Events`, `bitshares.instance.BlockchainInstance`

Notifications on Blockchain events.

### Parameters

- **accounts** (*list*) – Account names/ids to be notified about when changing
- **markets** (*list*) – Instances of `bitshares.market.Market` that identify markets to be monitored
- **objects** (*list*) – Object ids to be notified about when changed
- **on\_tx** (*fn*) – Callback that will be called for each transaction received
- **on\_block** (*fn*) – Callback that will be called for each block received
- **on\_account** (*fn*) – Callback that will be called for changes of the listed accounts
- **on\_market** (*fn*) – Callback that will be called for changes of the listed markets
- **blockchain\_instance** (`bitshares.bitshares.BitShares`) – BitShares instance

### Example

```
from pprint import pprint
from bitshares.notify import Notify
from bitshares.market import Market

notify = Notify(
    markets=["TEST:GOLD"],
    accounts=["xeroc"],
    on_market=print,
    on_account=print,
    on_block=print,
    on_tx=print
)
notify.listen()
```

**bitshares**

Alias for the specific blockchain.

**blockchain****chain**

Short form for blockchain (for the lazy)

**close()**

Cleanly close the Notify instance.

**define\_classes()**

Needs to define instance variables that provide classes

**get\_instance\_class()**

Should return the Chain instance class, e.g. *bitshares.BitShares*

**get\_market\_ids(markets)****classmethod inject(cls)****listen()**

This call initiates the listening/notification process.

It behaves similar to `run_forever()`.

**process\_account(message)**

This is used for processing of account Updates.

It will return instances of `:class:bitshares.account.AccountUpdate`

**process\_market(data)**

This method is used for post processing of market notifications. It will return instances of either.

- *bitshares.price.Order* or
- *bitshares.price.FilledOrder* or
- *bitshares.price.UpdateCallOrder*

Also possible are limit order updates (margin calls)

**reset\_subscriptions(accounts=None, markets=None, objects=None)**

Change the subscriptions of a running Notify instance.

**classmethod set\_shared\_blockchain\_instance(instance)**

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance(chaininstance)* – Chain instance

**classmethod** `set_shared_config (config)`

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance ()**

This method allows to set the current instance as default

**shared\_blockchain\_instance ()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

## bitshares.price module

**class** `bitshares.price.FilledOrder (order, **kwargs)`

Bases: `bitshares.price.Price`

This class inherits `bitshares.price.Price` but has the base and quote Amounts not only be used to represent the price (as a ratio of base and quote) but instead has those amounts represent the amounts of an actually filled order!

**Parameters** `blockchain_instance` (`bitshares.bitshares.BitShares`) – BitShares instance

---

**Note:** Instances of this class come with an additional `time` key that shows when the order has been filled!

---

**as\_base** (*base*)

Returns the price instance so that the base asset is *base*.

Note: This makes a copy of the object!

**as\_quote** (*quote*)

Returns the price instance so that the quote asset is *quote*.

Note: This makes a copy of the object!

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class()**

Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject(cls)**

**invert()**

Invert the price (e.g. go from USD/BTS into BTS/USD)

**items()** → a set-like object providing a view on D's items

**json()**

```
return { "base": self["base"].json(), "quote": self["quote"].json()
}
```

**keys()** → a set-like object providing a view on D's keys

**market**

Open the corresponding market.

**Returns** Instance of *bitshares.market.Market* for the corresponding pair of assets.

**pop(k[, d])** → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise *KeyError* is raised

**popitem()** → (k, v), remove and return some (key, value) pair as a 2-tuple; but raise *KeyError* if D is empty.

**classmethod set\_shared\_blockchain\_instance(instance)**

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters instance(chaininstance)** – Chain instance

**classmethod set\_shared\_config(config)**

This allows to set a config that will be used when calling *shared\_blockchain\_instance* and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**

This method will initialize *SharedInstance.instance* and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**symbols()**

**update([E], \*\*F)** → None. Update D from dict/iterable E and F.

If E is present and has a *.keys()* method, then does: for k in E: D[k] = E[k] If E is present and lacks a *.keys()* method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class bitshares.price.Order(\*args, \*\*kwargs)**

Bases: *bitshares.price.Price*

This class inherits *bitshares.price.Price* but has the *base* and *quote* Amounts not only be used to represent the price (as a ratio of base and quote) but instead has those amounts represent the amounts of an actual order!

**Parameters** `blockchain_instance` (`bitshares.bitshares.BitShares`) – BitShares instance

---

**Note:** If an order is marked as deleted, it will carry the ‘deleted’ key which is set to `True` and all other data be `None`.

---

**as\_base** (*base*)

Returns the price instance so that the base asset is *base*.

Note: This makes a copy of the object!

**as\_quote** (*quote*)

Returns the price instance so that the quote asset is *quote*.

Note: This makes a copy of the object!

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**chain**

Short form for blockchain (for the lazy)

**clear** () → `None`. Remove all items from `D`.

**copy** () → a shallow copy of `D`

**define\_classes** ()

Needs to define instance variables that provide classes

**for\_sale**

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. `bitshares.BitShares`

**classmethod inject** (*cls*)

**invert** ()

Invert the price (e.g. go from `USD/BTS` into `BTS/USD`)

**items** () → a set-like object providing a view on `D`’s items

**json** ()

```
return { "base": self["base"].json(), "quote": self["quote"].json()
}
```

**keys** () → a set-like object providing a view on `D`’s keys

**market**

Open the corresponding market.

**Returns** Instance of `bitshares.market.Market` for the corresponding pair of assets.

**pop** ( $k, d$ )  $\rightarrow v$ , remove specified key and return the corresponding value.

If key is not found,  $d$  is returned if given, otherwise `KeyError` is raised

**popitem** ()  $\rightarrow (k, v)$ , remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if  $D$  is empty.

**price**

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**symbols** ()

**to\_buy**

**update** ( $[E]$ ,  $**F$ )  $\rightarrow$  None. Update  $D$  from dict/iterable  $E$  and  $F$ .

If  $E$  is present and has a `.keys()` method, then does: for  $k$  in  $E$ :  $D[k] = E[k]$  If  $E$  is present and lacks a `.keys()` method, then does: for  $k, v$  in  $E$ :  $D[k] = v$  In either case, this is followed by: for  $k$  in  $F$ :  $D[k] = F[k]$

**values** ()  $\rightarrow$  an object providing a view on  $D$ 's values

**class** `bitshares.price.Price` (\*args, \*\*kwargs)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.price.Price`

This class deals with all sorts of prices of any pair of assets to simplify dealing with the tuple:

```
(quote, base)

each being an instance of :class:`bitshares.amount.Amount`. The
amount themselves define the price.

.. note::

    The price (floating) is derived as ``base/quote``

:param list args: Allows to deal with different representations of a price
:param bitshares.asset.Asset base: Base asset
:param bitshares.asset.Asset quote: Quote asset
:param bitshares.bitshares.BitShares blockchain_instance: BitShares instance
:returns: All data required to represent a price
:rtype: dict

Way to obtain a proper instance:
```

(continues on next page)

(continued from previous page)

```

* ``args`` is a str with a price and two assets
* ``args`` can be a floating number and ``base`` and ``quote`` being
↳instances of :class:`bitshares.asset.Asset`
* ``args`` can be a floating number and ``base`` and ``quote`` being
↳instances of ``str``
* ``args`` can be dict with keys ``price``, ``base``, and ``quote``
↳(*graphene balances*)
* ``args`` can be dict with keys ``base`` and ``quote``
* ``args`` can be dict with key ``receives`` (filled orders)
* ``args`` being a list of ``[quote, base]`` both being instances of
↳:class:`bitshares.amount.Amount`
* ``args`` being a list of ``[quote, base]`` both being instances of ``str``
↳(``amount symbol``)
* ``base`` and ``quote`` being instances of :class:`bitshares.asset.Amount`

```

This allows instantiations like:

```

* ``Price("0.315 USD/BTS")``
* ``Price(0.315, base="USD", quote="BTS")``
* ``Price(0.315, base=Asset("USD"), quote=Asset("BTS"))``
* ``Price({"base": {"amount": 1, "asset_id": "1.3.0"}, "quote": {"amount": 10,
↳"asset_id": "1.3.106"}})``
* ``Price({"receives": {"amount": 1, "asset_id": "1.3.0"}, "pays": {"amount": 10,
↳"asset_id": "1.3.106"}}, base_asset=Asset("1.3.0"))``
* ``Price(quote="10 GOLD", base="1 USD")``
* ``Price("10 GOLD", "1 USD")``
* ``Price(Amount("10 GOLD"), Amount("1 USD"))``
* ``Price(1.0, "USD/GOLD")``

```

Instances of this class can be used in regular mathematical expressions  
(``+-\*/%``) such as:

```
.. code-block:: python
```

```

>>> from bitshares.price import Price
>>> Price("0.3314 USD/BTS") * 2
0.662600000 USD/BTS

```

#### **as\_base** (*base*)

Returns the price instance so that the base asset is base.

Note: This makes a copy of the object!

#### **as\_quote** (*quote*)

Returns the price instance so that the quote asset is quote.

Note: This makes a copy of the object!

#### **bitshares**

Alias for the specific blockchain.

#### **blockchain**

#### **blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

#### **chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**copy()** → a shallow copy of D

**define\_classes()**

Needs to define instance variables that provide classes

**fromkeys()**

Create a new dictionary with keys from iterable and values set to value.

**get()**

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class()**

Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject(cls)**

**invert()**

Invert the price (e.g. go from USD/BTS into BTS/USD)

**items()** → a set-like object providing a view on D's items

**json()**

```
return { "base": self["base"].json(), "quote": self["quote"].json()
}
```

**keys()** → a set-like object providing a view on D's keys

**market**

Open the corresponding market.

**Returns** Instance of *bitshares.market.Market* for the corresponding pair of assets.

**pop(k[, d])** → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise *KeyError* is raised

**popitem()** → (k, v), remove and return some (key, value) pair as a

2-tuple; but raise *KeyError* if D is empty.

**classmethod set\_shared\_blockchain\_instance(instance)**

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters instance(chaininstance)** – Chain instance

**classmethod set\_shared\_config(config)**

This allows to set a config that will be used when calling *shared\_blockchain\_instance* and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**

This method will initialize *SharedInstance.instance* and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**symbols()**

**update([E], \*\*F)** → None. Update D from dict/iterable E and F.

If E is present and has a *.keys()* method, then does: for k in E: D[k] = E[k] If E is present and lacks a *.keys()* method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]



**values** () → an object providing a view on D's values

**class** `bitshares.price.PriceFeed(*args, **kwargs)`

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.price.PriceFeed`

This class is used to represent a price feed consisting of.

- a witness,
- a symbol,
- a core exchange rate,
- the maintenance collateral ratio,
- the max short squeeze ratio,
- a settlement price, and
- a date

**Parameters** `blockchain_instance` (`bitshares.bitshares.BitShares`) – BitShares instance

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. `bitshares.BitShares`

**classmethod inject** (*cls*)

**items** () → a set-like object providing a view on D's items

**keys** () → a set-like object providing a view on D's keys

**pop** (*k*, *d*) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a

2-tuple; but raise `KeyError` if D is empty.

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** `instance` (*chaininstance*) – Chain instance

**classmethod** `set_shared_config` (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** `bitshares.price.UpdateCallOrder` (*call*, *\*\*kwargs*)

Bases: `bitshares.price.Price`

This class inherits `bitshares.price.Price` but has the base and quote Amounts not only be used to represent the **call price** (as a ratio of base and quote).

**Parameters** `blockchain_instance` (`bitshares.bitshares.BitShares`) – BitShares instance

**as\_base** (*base*)

Returns the price instance so that the base asset is base.

Note: This makes a copy of the object!

**as\_quote** (*quote*)

Returns the price instance so that the quote asset is quote.

Note: This makes a copy of the object!

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()  
Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject** (cls)

**invert** ()  
Invert the price (e.g. go from USD/BTS into BTS/USD)

**items** () → a set-like object providing a view on D's items

**json** ()  

```

    return { "base": self["base"].json(), "quote": self["quote"].json()
    }

```

**keys** () → a set-like object providing a view on D's keys

**market**  
Open the corresponding market.  
**Returns** Instance of *bitshares.market.Market* for the corresponding pair of assets.

**pop** (k[, d]) → v, remove specified key and return the corresponding value.  
If key is not found, d is returned if given, otherwise *KeyError* is raised

**popitem** () → (k, v), remove and return some (key, value) pair as a 2-tuple; but raise *KeyError* if D is empty.

**classmethod set\_shared\_blockchain\_instance** (instance)  
This method allows us to override default instance for all users of *SharedInstance.instance*.  
**Parameters** *instance* (chaininstance) – Chain instance

**classmethod set\_shared\_config** (config)  
This allows to set a config that will be used when calling *shared\_blockchain\_instance* and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()  
This method allows to set the current instance as default

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()  
This method will initialize *SharedInstance.instance* and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**symbols** ()

**update** ([E], \*\*F) → None. Update D from dict/iterable E and F.  
If E is present and has a *.keys()* method, then does: for k in E: D[k] = E[k] If E is present and lacks a *.keys()* method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**bitshares.proposal module****class** bitshares.proposal.**Proposal** (\*args, \*\*kwargs)Bases: *bitshares.instance.BlockchainInstance*, *bitshares.proposal.Proposal*

Read data about a Proposal Balance in the chain.

**Parameters**

- **id** (*str*) – Id of the proposal
- **blockchain\_instance** (*bitshares*) – BitShares() instance to use when accessing a RPC

**bitshares**

Alias for the specific blockchain.

**blockchain****blockchain\_instance\_class**alias of *bitshares.instance.BlockchainInstance***classmethod** **cache\_object** (*data*, *key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.**classmethod** **clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D**define\_classes** ()

Needs to define instance variables that provide classes

**expiration****fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()Should return the Chain instance class, e.g. *bitshares.BitShares***getfromcache** (*id*)

Get an element from the cache explicitly

**identifier** = None**incached** (*id*)

Is an element cached?

**classmethod** **inject** (*cls*)**is\_in\_review****items** ()

This overwrites items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static objectid\_valid(*i*)**

Test if a string looks like a regular object id of the form::

```
xxxx.yyyyyy.zzzz
```

with those being numbers.

**perform\_id\_tests = True**

**pop(*k*, *d*)** → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

**popitem()** → (*k*, *v*), remove and return some (key, value) pair as a 2-tuple; but raise `KeyError` if *D* is empty.

**proposed\_operations**

**proposer**

Return the proposer of the proposal if available in the backend, else returns `None`

**refresh()**

**review\_period**

**static set\_cache\_store(*klass*, \**args*, \*\**kwargs*)**

**classmethod set\_shared\_blockchain\_instance(*instance*)**

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters *instance* (*chaininstance*)** – Chain instance

**classmethod set\_shared\_config(*config*)**

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id = 1**

**store(*data*, *key*='id')**

Cache the list

**Parameters *data* (*list*)** – List of objects to cache

**test\_valid\_objectid(*i*)**

Alias for `objectid_valid`

**testid(*id*)**

In contrast to validity, this method tests if the objectid matches the `type_id` provided in `self.type_id` or `self.type_ids`

**type\_id = None**

**type\_ids = []**

**update** (*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** `bitshares.proposal.Proposals` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.proposal.Proposals`

Obtain a list of pending proposals for an account.

#### Parameters

- **account** (*str*) – Account name
- **blockchain\_instance** (*bitshares*) – BitShares() instance to use when accessing a RPC

**append** ()

Append object to the end of the list.

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**cache** (*key*)

(legacy) store the current object with key *key*.

**classmethod** **cache\_objects** (*data*, *key=None*)

This classmethod allows to feed multiple objects into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** ()

Remove all items from list.

**classmethod** **clear\_cache** ()

Clear/Reset the entire Cache

**copy** ()

Return a shallow copy of the list.

**count** ()

Return number of occurrences of value.

**define\_classes** ()

Needs to define instance variables that provide classes

**extend** ()

Extend list by appending elements from the iterable.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. `bitshares.BitShares`

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier** = None

**incached** (*id*)  
Is an element cached?

**index** ()  
Return first index of value.  
Raises `ValueError` if the value is not present.

**classmethod inject** (*cls*)

**insert** ()  
Insert object before index.

**items** ()  
This overwrites `items()` so that `refresh()` is called if the object is not already fetched

**pop** ()  
Remove and return item at index (default last).  
Raises `IndexError` if list is empty or index is out of range.

**refresh** (*\*args, \*\*kwargs*)  
Interface that needs to be implemented. This method is called when an object is requested that has not yet been fetched/stored

**remove** ()  
Remove first occurrence of value.  
Raises `ValueError` if the value is not present.

**reverse** ()  
Reverse *IN PLACE*.

**static set\_cache\_store** (*klass, \*args, \*\*kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)  
This method allows us to override default instance for all users of `SharedInstance.instance`.  
  
**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)  
This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()  
This method allows to set the current instance as default

**shared\_blockchain\_instance** ()  
This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**sort** ()  
Stable sort *IN PLACE*.

**store** (*data, key=None, \*args, \*\*kwargs*)  
Cache the list  
  
**Parameters** *data* (*list*) – List of objects to cache

## bitshares.storage module

`bitshares.storage.get_default_config_store (*args, **kwargs)`

```
bitshares.storage.get_default_key_store (config, *args, **kwargs)
```

## bitshares.transactionbuilder module

```
class bitshares.transactionbuilder.ProposalBuilder (*args, **kwargs)
```

Bases: *bitshares.instance.BlockchainInstance*, *bitshares.transactionbuilder.ProposalBuilder*

Proposal Builder allows us to construct an independent Proposal that may later be added to an instance of TransactionBuilder.

### Parameters

- **proposer** (*str*) – Account name of the proposing user
- **proposal\_expiration** (*int*) – Number seconds until the proposal is supposed to expire
- **proposal\_review** (*int*) – Number of seconds for review of the proposal
- **transactionbuilder.TransactionBuilder** – Specify your own instance of transaction builder (optional)
- **blockchain\_instance** (*instance*) – Blockchain instance

```
appendOps (ops, append_to=None)
```

Append op(s) to the transaction builder

**Parameters** **ops** (*list*) – One or a list of operations

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of *bitshares.instance.BlockchainInstance*

**broadcast** ()

**chain**

Short form for blockchain (for the lazy)

**define\_classes** ()

Needs to define instance variables that provide classes

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**get\_parent** ()

This allows to refer to the actual parent of the Proposal

**get\_raw** ()

Returns an instance of base “Operations” for further processing

**classmethod inject** (*cls*)

**is\_empty** ()

**json** ()

Return the json formatted version of this proposal

**list\_operations** ()



**set\_expiration** (*p*)

**set\_parent** (*p*)

**set\_proposer** (*p*)

**set\_review** (*p*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**class** `bitshares.transactionbuilder.TransactionBuilder` (*\*args, \*\*kwargs*)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.transactionbuilder.TransactionBuilder`

This class simplifies the creation of transactions by adding operations and signers.

**addSigningInformation** (*account, permission*)

This is a private method that adds side information to a unsigned/partial transaction in order to simplify later signing (e.g. for multisig or coldstorage)

FIXME: Does not work with owner keys!

**add\_required\_fees** (*ops, asset\_id='1.3.0'*)

Auxiliary method to obtain the required fees for a set of operations. Requires a websocket connection to a witness node!

**appendMissingSignatures** ()

Store which accounts/keys are supposed to sign the transaction

This method is used for an offline-signer!

**appendOps** (*ops, append\_to=None*)

Append op(s) to the transaction builder

**Parameters** *ops* (*list*) – One or a list of operations

**appendSigner** (*accounts, permission*)

Try to obtain the wif key from the wallet by telling which account and permission is supposed to sign the transaction

**Parameters**

- **accounts** (*str, list, tuple, set*) – accounts to sign transaction with
- **permission** (*str*) – type of permission, e.g. “active”, “owner” etc

**appendWif** (*wif*)

Add a wif that should be used for signing of the transaction.

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of *bitshares.instance.BlockchainInstance*

**broadcast()**

Broadcast a transaction to the blockchain network

**Parameters** **tx** (*tx*) – Signed transaction to broadcast

**chain**

Short form for blockchain (for the lazy)

**clear()**

Clear the transaction builder and start from scratch

**constructTx()**

Construct the actual transaction and store it in the class's dict store

**copy()** → a shallow copy of D

**define\_classes()**

Needs to define instance variables that provide classes

**fromkeys()**

Create a new dictionary with keys from iterable and values set to value.

**get()**

Return the value for key if key is in the dictionary, else default.

**get\_block\_params** (*use\_head\_block=False*)

Auxiliary method to obtain *ref\_block\_num* and *ref\_block\_prefix*. Requires a websocket connection to a witness node!

**get\_instance\_class()**

Should return the Chain instance class, e.g. *bitshares.BitShares*

**get\_parent()**

TransactionBuilders don't have parents, they are their own parent

**classmethod inject** (*cls*)

**is\_empty()**

**items()** → a set-like object providing a view on D's items

**json()**

Show the transaction as plain json

**keys()** → a set-like object providing a view on D's keys

**list\_operations()**

**permission\_types** = ['active', 'owner']

**pop** (*k* [, *d*]) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise *KeyError* is raised

**popitem()** → (*k*, *v*), remove and return some (key, value) pair as a

2-tuple; but raise *KeyError* if D is empty.

**set\_expiration** (*p*)

**set\_fee\_asset** (*fee\_asset*)

Set asset to fee

**classmethod `set_shared_blockchain_instance`** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod `set_shared_config`** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**`set_shared_instance`** ()

This method allows to set the current instance as default

**`setdefault`** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**`shared_blockchain_instance`** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**`sign`** ()

Sign a provided transaction with the provided key(s)

**Parameters**

- **`tx`** (*dict*) – The transaction to be signed and returned
- **`wifs`** (*string*) – One or many wif keys to use for signing a transaction. If not present, the keys will be loaded from the wallet as defined in “missing\_signatures” key of the transactions.

**`update`** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**`values`** () → an object providing a view on D’s values

**`verify_authority`** ()

Verify the authority of the signed transaction

## bitshares.utils module

## bitshares.vesting module

**class** `bitshares.vesting.Vesting` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.vesting.Vesting`

Read data about a Vesting Balance in the chain.

**Parameters**

- **`id`** (*str*) – Id of the vesting balance
- **`blockchain_instance`** (*bitshares*) – BitShares() instance to use when accessing a RPC

**`account`**

**`bitshares`**

Alias for the specific blockchain.

**`blockchain`**

**blockchain\_instance\_class**  
alias of *bitshares.instance.BlockchainInstance*

**classmethod cache\_object** (*data, key=None*)  
This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**  
Short form for blockchain (for the lazy)

**claim** (*amount=None*)

**claimable**

**clear** () → None. Remove all items from D.

**classmethod clear\_cache** ()  
Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()  
Needs to define instance variables that provide classes

**fromkeys** ()  
Create a new dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()  
Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)  
Get an element from the cache explicitly

**identifier = None**

**incached** (*id*)  
Is an element cached?

**classmethod inject** (*cls*)

**items** ()  
This overwrites items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static objectid\_valid** (*i*)  
Test if a string looks like a regular object id of the form::

`xxxx.yyzzz.zzzz`

with those being numbers.

**perform\_id\_tests = True**

**pop** (*k[, d]*) → v, remove specified key and return the corresponding value.  
If key is not found, d is returned if given, otherwise *KeyError* is raised

**popitem** () → (k, v), remove and return some (key, value) pair as a 2-tuple; but raise *KeyError* if D is empty.

**refresh** ()

**static set\_cache\_store** (*klass, \*args, \*\*kwargs*)

**classmethod `set_shared_blockchain_instance`** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod `set_shared_config`** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**`set_shared_instance`** ()

This method allows to set the current instance as default

**`setdefault`** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**`shared_blockchain_instance`** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**`space_id`** = 1

**`store`** (*data*, *key*='id')

Cache the list

**Parameters** *data* (*list*) – List of objects to cache

**`test_valid_objectid`** (*i*)

Alias for `objectid_valid`

**`testid`** (*id*)

In contrast to validity, this method tests if the `objectid` matches the `type_id` provided in `self.type_id` or `self.type_ids`

**`type_id`** = None

**`type_ids`** = []

**`update`** ([*E*], **`**F`**) → None. Update *D* from dict/iterable *E* and *F*.

If *E* is present and has a `.keys()` method, then does: for *k* in *E*: *D*[*k*] = *E*[*k*] If *E* is present and lacks a `.keys()` method, then does: for *k*, *v* in *E*: *D*[*k*] = *v* In either case, this is followed by: for *k* in *F*: *D*[*k*] = *F*[*k*]

**`values`** () → an object providing a view on *D*'s values

## bitshares.wallet module

**class** `bitshares.wallet.Wallet` (*\*args*, **`**kwargs`**)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.wallet.Wallet`

**`addPrivateKey`** (*wif*)

Add a private key to the wallet database

**`bitshares`**

Alias for the specific blockchain.

**`blockchain`**

**`blockchain_instance_class`**

alias of `bitshares.instance.BlockchainInstance`

**`chain`**

Short form for `blockchain` (for the lazy)

**changePassphrase** (*new\_pwd*)  
Change the passphrase for the wallet database

**create** (*pwd*)  
Alias for newWallet()

**created** ()  
Do we have a wallet database already?

**define\_classes** ()  
Needs to define instance variables that provide classes

**getAccountFromPrivateKey** (*wif*)  
Obtain account name from private key

**getAccountFromPublicKey** (*pub*)  
Obtain the first account name from public key

**getAccounts** ()  
Return all accounts installed in the wallet database

**getAccountsFromPublicKey** (*pub*)  
Obtain all accounts associated with a public key

**getActiveKeyForAccount** (*name*)  
Obtain owner Active Key for an account from the wallet database

**getAllAccounts** (*pub*)  
Get the account data for a public key (all accounts found for this public key)

**getKeyType** (*account, pub*)  
Get key type

**getMemoKeyForAccount** (*name*)  
Obtain owner Memo Key for an account from the wallet database

**getOwnerKeyForAccount** (*name*)  
Obtain owner Private Key for an account from the wallet database

**getPrivateKeyForPublicKey** (*pub*)  
Obtain the private key for a given public key

**Parameters** *pub* (*str*) – Public Key

**getPublicKeys** (*current=False*)  
Return all installed public keys

**Parameters** *current* (*bool*) – If true, returns only keys for currently connected blockchain

**get\_instance\_class** ()  
Should return the Chain instance class, e.g. *bitshares.BitShares*

**classmethod inject** (*cls*)

**is\_encrypted** ()  
Is the key store encrypted?

**lock** ()  
Lock the wallet database

**locked** ()  
Is the wallet database locked?

**newWallet** (*pwd*)  
Create a new wallet database

**prefix**

**privatekey** (*key*)

**publickey\_from\_wif** (*wif*)

**removeAccount** (*account*)

Remove all keys associated with a given account

**removePrivateKeyFromPublicKey** (*pub*)

Remove a key from the wallet database

**rpc**

**setKeys** (*loadkeys*)

This method is strictly only for in memory keys that are passed to Wallet with the *keys* argument

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**unlock** (*pwd*)

Unlock the wallet database

**unlocked** ()

Is the wallet database unlocked?

**wipe** (*sure=False*)

## bitshares.witness module

**class** `bitshares.witness.Witness` (\*args, \*\*kwargs)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.witness.Witness`

Read data about a witness in the chain.

**Parameters**

- **account\_name** (*str*) – Name of the witness
- **blockchain\_instance** (*bitshares*) – BitShares() instance to use when accessing a RPC

**account**

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**classmethod** **cache\_object** (*data*, *key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod** **clear\_cache** ()

Clear/Reset the entire Cache

**copy** () → a shallow copy of D

**define\_classes** ()

Needs to define instance variables that provide classes

**fromkeys** ()

Create a new dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class** ()

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)

Get an element from the cache explicitly

**identifier** = None

**incached** (*id*)

Is an element cached?

**classmethod** **inject** (*cls*)

**is\_active**

**items** ()

This overwrites items() so that refresh() is called if the object is not already fetched

**keys** () → a set-like object providing a view on D's keys

**static** **objectid\_valid** (*i*)

Test if a string looks like a regular object id of the form::

`xxxx.yyyyyy.zzzz`

with those being numbers.

**perform\_id\_tests** = True

**pop** (*k*[, *d*]) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise *KeyError* is raised

**popitem** () → (*k*, *v*), remove and return some (key, value) pair as a

2-tuple; but raise *KeyError* if D is empty.

**refresh** ()

**static** **set\_cache\_store** (*klass*, *\*args*, *\*\*kwargs*)

**classmethod** **set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of *SharedInstance.instance*.

**Parameters** **instance** (*chaininstance*) – Chain instance



**classmethod** `set_shared_config (config)`

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance** ()

This method allows to set the current instance as default

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance** ()

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id** = 1

**store** (data, key='id')

Cache the list

**Parameters** `data (list)` – List of objects to cache

**test\_valid\_objectid** (i)

Alias for `objectid_valid`

**testid** (id)

In contrast to validity, this method tests if the objectid matches the type\_id provided in `self.type_id` or `self.type_ids`

**type\_id** = None

**type\_ids** = []

**update** ([E], \*\*F) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**weight**

**class** `bitshares.witness.Witnesses (*args, **kwargs)`

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.witness.Witnesses`

Obtain a list of **active** witnesses and the current schedule.

**Parameters**

- **only\_active** (bool) – (False) Only return witnesses that are actively producing blocks
- **blockchain\_instance** (bitshares) – BitShares() instance to use when accesing a RPC

**append** ()

Append object to the end of the list.

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**cache** (*key*)  
(legacy) store the current object with key *key*.

**classmethod cache\_objects** (*data, key=None*)  
This classmethod allows to feed multiple objects into the cache is is mostly used for testing

**chain**  
Short form for blockchain (for the lazy)

**clear** ()  
Remove all items from list.

**classmethod clear\_cache** ()  
Clear/Reset the entire Cache

**copy** ()  
Return a shallow copy of the list.

**count** ()  
Return number of occurrences of value.

**define\_classes** ()  
Needs to define instance variables that provide classes

**extend** ()  
Extend list by appending elements from the iterable.

**get\_instance\_class** ()  
Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache** (*id*)  
Get an element from the cache explicitly

**identifier = None**

**incached** (*id*)  
Is an element cached?

**index** ()  
Return first index of value.  
  
Raises *ValueError* if the value is not present.

**classmethod inject** (*cls*)

**insert** ()  
Insert object before index.

**items** ()  
This overwrites items() so that refresh() is called if the object is not already fetched

**pop** ()  
Remove and return item at index (default last).  
  
Raises *IndexError* if list is empty or index is out of range.

**refresh** (*\*args, \*\*kwargs*)  
Interface that needs to be implemented. This method is called when an object is requested that has not yet been fetched/stored

**remove** ()  
Remove first occurrence of value.  
  
Raises *ValueError* if the value is not present.

**reverse()**

Reverse *IN PLACE*.

**static set\_cache\_store** (*klass*, \**args*, \*\**kwargs*)

**classmethod set\_shared\_blockchain\_instance** (*instance*)

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters** *instance* (*chaininstance*) – Chain instance

**classmethod set\_shared\_config** (*config*)

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**sort()**

Stable sort *IN PLACE*.

**store** (*data*, *key=None*, \**args*, \*\**kwargs*)

Cache the list

**Parameters** *data* (*list*) – List of objects to cache

## bitshares.worker module

**class** `bitshares.worker.Worker` (\**args*, \*\**kwargs*)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.worker.Worker`

Read data about a worker in the chain.

**Parameters**

- **id** (*str*) – id of the worker
- **blockchain\_instance** (*bitshares*) – BitShares() instance to use when accesing a RPC

**account**

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**classmethod cache\_object** (*data*, *key=None*)

This classmethod allows to feed an object into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** () → None. Remove all items from D.

**classmethod clear\_cache** ()

Clear/Reset the entire Cache

**copy()** → a shallow copy of D

**define\_classes()**

Needs to define instance variables that provide classes

**fromkeys()**

Create a new dictionary with keys from iterable and values set to value.

**get()**

Return the value for key if key is in the dictionary, else default.

**get\_instance\_class()**

Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache(id)**

Get an element from the cache explicitly

**identifier = None**

**incached(id)**

Is an element cached?

**classmethod inject(cls)**

**items()**

This overwrites items() so that refresh() is called if the object is not already fetched

**keys()** → a set-like object providing a view on D's keys

**static objectid\_valid(i)**

Test if a string looks like a regular object id of the form::

`xxxx.yyyyy.zzzz`

with those being numbers.

**perform\_id\_tests = True**

**pop(k[, d])** → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise `KeyError` is raised

**popitem()** → (k, v), remove and return some (key, value) pair as a

2-tuple; but raise `KeyError` if D is empty.

**post\_format()**

**refresh()**

**static set\_cache\_store(klass, \*args, \*\*kwargs)**

**classmethod set\_shared\_blockchain\_instance(instance)**

This method allows us to override default instance for all users of `SharedInstance.instance`.

**Parameters instance(chaininstance)** – Chain instance

**classmethod set\_shared\_config(config)**

This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**space\_id = 1**

**store** (*data*, *key='id'*)

Cache the list

**Parameters** *data* (*list*) – List of objects to cache

**test\_valid\_objectid** (*i*)

Alias for `objectid_valid`

**testid** (*id*)

In contrast to validity, this method tests if the objectid matches the type\_id provided in `self.type_id` or `self.type_ids`

**type\_id = None**

**type\_ids = []**

**update** (*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** `bitshares.worker.Workers` (*\*args*, *\*\*kwargs*)

Bases: `bitshares.instance.BlockchainInstance`, `bitshares.worker.Workers`

Obtain a list of workers for an account.

**Parameters**

- **account\_name/id** (*str*) – Name/id of the account (optional)
- **blockchain\_instance** (*bitshares*) – BitShares() instance to use when accessing a RPC

**append** ()

Append object to the end of the list.

**bitshares**

Alias for the specific blockchain.

**blockchain**

**blockchain\_instance\_class**

alias of `bitshares.instance.BlockchainInstance`

**cache** (*key*)

(legacy) store the current object with key *key*.

**classmethod** **cache\_objects** (*data*, *key=None*)

This classmethod allows to feed multiple objects into the cache is is mostly used for testing

**chain**

Short form for blockchain (for the lazy)

**clear** ()

Remove all items from list.

**classmethod** **clear\_cache** ()

Clear/Reset the entire Cache

**copy()**  
Return a shallow copy of the list.

**count()**  
Return number of occurrences of value.

**define\_classes()**  
Needs to define instance variables that provide classes

**extend()**  
Extend list by appending elements from the iterable.

**get\_instance\_class()**  
Should return the Chain instance class, e.g. *bitshares.BitShares*

**getfromcache(id)**  
Get an element from the cache explicitly

**identifier = None**

**incached(id)**  
Is an element cached?

**index()**  
Return first index of value.  
  
Raises `ValueError` if the value is not present.

**classmethod inject(cls)**

**insert()**  
Insert object before index.

**items()**  
This overwrites `items()` so that `refresh()` is called if the object is not already fetched

**pop()**  
Remove and return item at index (default last).  
  
Raises `IndexError` if list is empty or index is out of range.

**refresh(\*args, \*\*kwargs)**  
Interface that needs to be implemented. This method is called when an object is requested that has not yet been fetched/stored

**remove()**  
Remove first occurrence of value.  
  
Raises `ValueError` if the value is not present.

**reverse()**  
Reverse *IN PLACE*.

**static set\_cache\_store(klass, \*args, \*\*kwargs)**

**classmethod set\_shared\_blockchain\_instance(instance)**  
This method allows us to override default instance for all users of `SharedInstance.instance`.  
  
**Parameters** `instance(chaininstance)` – Chain instance

**classmethod set\_shared\_config(config)**  
This allows to set a config that will be used when calling `shared_blockchain_instance` and allows to define the configuration without requiring to actually create an instance

**set\_shared\_instance()**

This method allows to set the current instance as default

**shared\_blockchain\_instance()**

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default instance that can be reused by multiple classes.

**sort()**

Stable sort *IN PLACE*.

**store(data, key=None, \*args, \*\*kwargs)**

Cache the list

**Parameters** `data` (*list*) – List of objects to cache

## Module contents

## 5.2 bitsharesbase

### 5.2.1 bitsharesbase package

#### Submodules

#### bitsharesbase.account module

**class** `bitsharesbase.account.Address` (*address, prefix=None*)

Bases: `graphenebase.account.Address`

Address class.

This class serves as an address representation for Public Keys.

#### Parameters

- **address** (*str*) – Base58 encoded address (defaults to None)
- **pubkey** (*str*) – Base58 encoded pubkey (defaults to None)
- **prefix** (*str*) – Network prefix (defaults to BTS)

Example:

```
Address("BTSFN9r6VYzBK8EKtMewfNbfiGCr56pHDBFi")
```

**classmethod** `from_pubkey` (*pubkey, compressed=True, version=56, prefix=None*)

Load an address provided the public key.

Version: 56 => PTS

**prefix** = 'BTS'

**set\_prefix** (*prefix*)

**class** `bitsharesbase.account.BrainKey` (*brainkey=None, sequence=0, prefix=None*)

Bases: `graphenebase.account.BrainKey`

Brainkey implementation similar to the graphene-ui web-wallet.

#### Parameters

- **brainkey** (*str*) – Brain Key

- **sequence** (*int*) – Sequence number for consecutive keys

Keys in Graphene are derived from a seed brain key which is a string of 16 words out of a predefined dictionary with 49744 words. It is a simple single-chain key derivation scheme that is not compatible with BIP44 but easy to use.

Given the brain key, a private key is derived as:

```
privkey = SHA256(SHA512(brainkey + " " + sequence))
```

Incrementing the sequence number yields a new key that can be regenerated given the brain key.

**get\_blind\_private()**

Derive private key from the brain key (and no sequence number)

**get\_brainkey()**

Return brain key of this instance

**get\_private()**

Derive private key from the brain key and the current sequence number

**get\_private\_key()**

**get\_public()**

**get\_public\_key()**

**next\_sequence()**

Increment the sequence number by 1

**normalize** (*brainkey*)

Correct formatting with single whitespace syntax and no trailing space

**prefix** = 'BTS'

**set\_prefix** (*prefix*)

**static suggest()**

Suggest a new random brain key. Randomness is provided by the operating system using `os.urandom()`.

**class** bitsharesbase.account.**PasswordKey** (*account, password, role='active', prefix=None*)

Bases: graphenebase.account.PasswordKey

This class derives a private key given the account name, the role and a password.

It leverages the technology of Brainkeys and allows people to have a secure private key by providing a passphrase only.

**get\_private()**

Derive private key from the brain key and the current sequence number

**get\_private\_key()**

**get\_public()**

**get\_public\_key()**

**prefix** = 'BTS'

**set\_prefix** (*prefix*)

**class** bitsharesbase.account.**PrivateKey** (*wif=None, prefix=None*)

Bases: graphenebase.account.PrivateKey

Derives the compressed and uncompressed public keys and constructs two instances of `PublicKey`:



**Parameters**

- **wif** (*str*) – Base58check-encoded wif key
- **prefix** (*str*) – Network prefix (defaults to BTS)

Example::

```
PrivateKey("5HqUkGuo62BfcJU5vNhTXKJRXuUi9QSE6jp8C3uBJ2BVHtB8WSd")
```

Compressed vs. Uncompressed:

- **PrivateKey("w-i-f").pubkey**: Instance of `PublicKey` using compressed key.
- **PrivateKey("w-i-f").pubkey.address**: Instance of `Address` using compressed key.
- **PrivateKey("w-i-f").uncompressed**: Instance of `PublicKey` using uncompressed key.
- **PrivateKey("w-i-f").uncompressed.address**: Instance of `Address` using uncompressed key.

**address**

**bitcoin**

**child** (*offset256*)

Derive new private key from this key and a sha256 “offset”

**compressed**

**derive\_from\_seed** (*offset*)

Derive private key using “generate\_from\_seed” method. Here, the key itself serves as a *seed*, and *offset* is expected to be a sha256 digest.

**derive\_private\_key** (*sequence*)

Derive new private key from this private key and an arbitrary sequence number

**get\_secret** ()

Get sha256 digest of the wif key.

**prefix** = 'BTS'

**pubkey**

**set\_prefix** (*prefix*)

**uncompressed**

**class** bitsharesbase.account.**PublicKey** (*pk*, *prefix=None*)

Bases: graphenebase.account.PublicKey

This class deals with Public Keys and inherits Address.

**Parameters**

- **pk** (*str*) – Base58 encoded public key
- **prefix** (*str*) – Network prefix (defaults to BTS)

Example::

```
PublicKey("BTS6UtYWWs3rkZGV8JA86qrgkG6tyFksgECefKE1MiH4HkLD8PFGL")
```

**Note:** By default, graphene-based networks deal with **compressed** public keys. If an **uncompressed** key is required, the method `unCompressed` can be used:

```
PublicKey("xxxxxx").unCompressed()
```

**add** (*digest256*)  
Derive new public key from this key and a sha256 “digest”

**address**  
Obtain a GrapheneAddress from a public key

**child** (*offset256*)  
Derive new public key from this key and a sha256 “offset”

**compressed** ()  
returns the compressed key

**compressed\_key**

**classmethod from\_privkey** (*privkey, prefix=None*)  
Derive uncompressed public key

**point** ()  
Return the point for the public key

**prefix** = 'BTS'

**pubkey**

**set\_prefix** (*prefix*)

**unCompressed** ()  
Alias for self.uncompressed() - LEGACY

**uncompressed** ()  
Derive uncompressed key

### bitsharesbase.asset\_permissions module

```
bitsharesbase.asset_permissions.force_flag(perms, flags)
bitsharesbase.asset_permissions.test_permissions(perms, flags)
bitsharesbase.asset_permissions.todict(number)
bitsharesbase.asset_permissions.toint(permissions)
```

### bitsharesbase.bip38 module

**bitsharesbase.bip38.decrypt** (*encrypted\_privkey, passphrase*)  
BIP0038 non-ec-multiply decryption. Returns WIF privkey.

#### Parameters

- **encrypted\_privkey** (*Base58*) – Private key
- **passphrase** (*str*) – UTF-8 encoded passphrase for decryption

**Returns** BIP0038 non-ec-multiply decrypted key

**Return type** Base58

**Raises** **SaltException** – if checksum verification failed (e.g. wrong password)

`bitsharesbase.bip38.encrypt(privkey, passphrase)`

BIP0038 non-ec-multiply encryption. Returns BIP0038 encrypted privkey.

**Parameters**

- **privkey** (*Base58*) – Private key
- **passphrase** (*str*) – UTF-8 encoded passphrase for encryption

**Returns** BIP0038 non-ec-multiply encrypted wif key

**Return type** Base58

## bitsharesbase.chains module

## bitsharesbase.memo module

## bitsharesbase.objects module

**class** `bitsharesbase.objects.AccountCreateExtensions(*args, **kwargs)`

Bases: `bitsharesbase.objects.Extension`

**class** `Buyback_options(*args, **kwargs)`

Bases: `graphenebase.objects.GrapheneObject`

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**

Read data explicitly (backwards compatibility)

**fromkeys** ()

Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()

Move an existing element to the end (or beginning if last is false).

Raise `KeyError` if the element does not exist.

**pop** (*k*, *d*) → v, remove specified key and return the corresponding

value. If key is not found, d is returned if given, otherwise `KeyError` is raised.

**popitem** ()

Remove and return a (key, value) pair from the dictionary.

Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** (*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** **Null\_ext** (*\*args*, *\*\*kwargs*)

Bases: `graphenebase.objects.GrapheneObject`

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**

Read data explicitly (backwards compatibility)

**fromkeys** ()

Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()

Move an existing element to the end (or beginning if last is false).

Raise `KeyError` if the element does not exist.

**pop** (*k*, *d*) → v, remove specified key and return the corresponding

value. If key is not found, d is returned if given, otherwise `KeyError` is raised.

**popitem** ()

Remove and return a (key, value) pair from the dictionary.

Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** (*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**sorted\_options** = [**'null\_ext'**, **<class 'bitsharesbase.objects.AccountCreateExtensions.N**

**bitsharesbase.objects.AccountId** (*asset*)

**class** **bitsharesbase.objects.AccountOptions** (*\*args*, *\*\*kwargs*)

Bases: `graphenebase.objects.GrapheneObject`

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(*k*, *d*)** → *v*, remove specified key and return the corresponding value. If key is not found, *d* is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update([*E*], \*\**F*)** → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** bitsharesbase.objects.**AssertPredicate**(*o*)  
Bases: graphenebase.types.Static\_variant

bitsharesbase.objects.**AssetId**(*asset*)

**class** bitsharesbase.objects.**AssetOptions**(*\*args*, *\*\*kwargs*)  
Bases: graphenebase.objects.GrapheneObject

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**

Move an existing element to the end (or beginning if last is false).

Raise `KeyError` if the element does not exist.

**pop**( $k, d$ )  $\rightarrow v$ , remove specified key and return the corresponding value. If key is not found,  $d$  is returned if given, otherwise `KeyError` is raised.

**popitem()**

Remove and return a (key, value) pair from the dictionary.

Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**toJson()**

**update**( $[E], **F$ )  $\rightarrow$  None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for  $k$  in E:  $D[k] = E[k]$  If E is present and lacks a `.keys()` method, then does: for  $k, v$  in E:  $D[k] = v$  In either case, this is followed by: for  $k$  in F:  $D[k] = F[k]$

**values()**  $\rightarrow$  an object providing a view on D's values

**class** `bitsharesbase.objects.BitAssetOptions` (*\*args, \*\*kwargs*)

Bases: `graphenebase.objects.GrapheneObject`

**clear()**  $\rightarrow$  None. Remove all items from od.

**copy()**  $\rightarrow$  a shallow copy of od

**data**

Read data explicitly (backwards compatibility)

**fromkeys()**

Create a new ordered dictionary with keys from iterable and values set to value.

**get()**

Return the value for key if key is in the dictionary, else default.

**items()**  $\rightarrow$  a set-like object providing a view on D's items

**json()**

**keys()**  $\rightarrow$  a set-like object providing a view on D's keys

**move\_to\_end()**

Move an existing element to the end (or beginning if last is false).

Raise `KeyError` if the element does not exist.

**pop**( $k, d$ )  $\rightarrow v$ , remove specified key and return the corresponding value. If key is not found,  $d$  is returned if given, otherwise `KeyError` is raised.

**popitem()**

Remove and return a (key, value) pair from the dictionary.

Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**toJson()**

**update** (*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.  
 If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.objects.**CallOrderExtension** (\*args, \*\*kwargs)  
 Bases: *bitsharesbase.objects.Extension*

**sorted\_options** = [('target\_collateral\_ratio', <function CallOrderExtension.targetCollateralRatio>)  
**targetCollateralRatio** ()

**class** bitsharesbase.objects.**Extension** (\*args, \*\*kwargs)  
 Bases: graphenebase.types.Array

**class** bitsharesbase.objects.**Memo** (\*args, \*\*kwargs)  
 Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
 Read data explicitly (backwards compatibility)

**fromkeys** ()  
 Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
 Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
 Move an existing element to the end (or beginning if last is false).  
 Raise KeyError if the element does not exist.

**pop** (*k*, *d*) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
 Remove and return a (key, value) pair from the dictionary.  
 Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
 Insert key with a value of default if key is not in the dictionary.  
 Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** (*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.  
 If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

```
class bitsharesbase.objects.ObjectId (object_str, type_verify=None)
    Bases: graphenebase.types.ObjectId
```

Need to overwrite a few attributes to load proper object\_types from bitshares.

```
    object_types = {'OBJECT_TYPE_COUNT': 19, 'account': 2, 'asset': 3, 'balance': 15, 'l
```

```
class bitsharesbase.objects.Operation (op, **kwargs)
    Bases: graphenebase.objects.Operation
```

Need to overwrite a few attributes to load proper operations from bitshares.

```
    append ()
        Append object to the end of the list.
```

```
    clear ()
        Remove all items from list.
```

```
    copy ()
        Return a shallow copy of the list.
```

```
    count ()
        Return number of occurrences of value.
```

```
    extend ()
        Extend list by appending elements from the iterable.
```

```
    fromlist = ['operations']
```

```
    getOperationIdForName (name)
```

```
    getOperationNameForId (i)
        Convert an operation id into the corresponding string
```

```
    id
```

```
    index ()
        Return first index of value.

        Raises ValueError if the value is not present.
```

```
    insert ()
        Insert object before index.
```

```
    json ()
```

```
    klass ()
```

```
    klass_name
```

```
    module = 'bitsharesbase.operations'
```

```
    op
```

```
    opId
```

```
    operation
```

```
    operations = {'account_create': 5, 'account_transfer': 9, 'account_update': 6, 'acc
```

```
    ops
```

```
    pop ()
        Remove and return item at index (default last).

        Raises IndexError if list is empty or index is out of range.
```



```

remove ()
    Remove first occurrence of value.

    Raises ValueError if the value is not present.

reverse ()
    Reverse IN PLACE.

set (**data)

sort ()
    Stable sort IN PLACE.

toJson ()

class bitsharesbase.objects.Permission (*args, **kwargs)
    Bases: graphenebase.objects.GrapheneObject

clear () → None. Remove all items from od.

copy () → a shallow copy of od

data
    Read data explicitly (backwards compatibility)

fromkeys ()
    Create a new ordered dictionary with keys from iterable and values set to value.

get ()
    Return the value for key if key is in the dictionary, else default.

items () → a set-like object providing a view on D's items

json ()

keys () → a set-like object providing a view on D's keys

move_to_end ()
    Move an existing element to the end (or beginning if last is false).

    Raise KeyError if the element does not exist.

pop (k[, d]) → v, remove specified key and return the corresponding
    value. If key is not found, d is returned if given, otherwise KeyError is raised.

popitem ()
    Remove and return a (key, value) pair from the dictionary.

    Pairs are returned in LIFO order if last is true or FIFO order if false.

setdefault ()
    Insert key with a value of default if key is not in the dictionary.

    Return the value for key if key is in the dictionary, else default.

toJson ()

update ([E], **F) → None. Update D from dict/iterable E and F.
    If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a
    .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values () → an object providing a view on D's values

class bitsharesbase.objects.Price (*args, **kwargs)
    Bases: graphenebase.objects.GrapheneObject

clear () → None. Remove all items from od.

```

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E], **F$ ) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.objects.**PriceFeed** (\*args, \*\*kwargs)  
Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end()**

Move an existing element to the end (or beginning if last is false).

Raise KeyError if the element does not exist.

**pop**( $k[d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem()**

Remove and return a (key, value) pair from the dictionary.

Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**toJson()**

**update**( $[E]$ ,  $**F$ ) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** bitsharesbase.objects.SpecialAuthority(o)

Bases: graphenebase.types.Static\_variant

**class** bitsharesbase.objects.Worker\_initializer(o)

Bases: graphenebase.types.Static\_variant

## bitsharesbase.objecttypes module

bitsharesbase.objecttypes.object\_type = {'OBJECT\_TYPE\_COUNT': 19, 'account': 2, 'asset':  
Object types for object ids

## bitsharesbase.operationids module

bitsharesbase.operationids.getOperationName(id: str)

This method returns the name representation of an operation given its value as used in the API.

bitsharesbase.operationids.getOperationNameForId(i)

Convert an operation id into the corresponding string.

bitsharesbase.operationids.ops = ['transfer', 'limit\_order\_create', 'limit\_order\_cancel',  
Operation ids

## bitsharesbase.operations module

**class** bitsharesbase.operations.Account\_create(\*args, \*\*kwargs)

Bases: graphenebase.objects.GrapheneObject

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**

Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E], **F$ ) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Account\_update** (\*args, \*\*kwargs)  
Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update([E], \*\*F)** → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** bitsharesbase.operations.**Account\_upgrade**(\*args, \*\*kwargs)  
Bases: graphenebase.objects.GrapheneObject

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(k[, d])** → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update([E], \*\*F)** → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

```
class bitsharesbase.operations.Account_whitelist (*args, **kwargs)
    Bases: graphenebase.objects.GrapheneObject

    black_listed = 2

    clear() → None. Remove all items from od.

    copy() → a shallow copy of od

    data
        Read data explicitly (backwards compatibility)

    fromkeys()
        Create a new ordered dictionary with keys from iterable and values set to value.

    get()
        Return the value for key if key is in the dictionary, else default.

    items() → a set-like object providing a view on D's items

    json()

    keys() → a set-like object providing a view on D's keys

    move_to_end()
        Move an existing element to the end (or beginning if last is false).

        Raise KeyError if the element does not exist.

    no_listing = 0

    pop(k[, d]) → v, remove specified key and return the corresponding
        value. If key is not found, d is returned if given, otherwise KeyError is raised.

    popitem()
        Remove and return a (key, value) pair from the dictionary.

        Pairs are returned in LIFO order if last is true or FIFO order if false.

    setdefault()
        Insert key with a value of default if key is not in the dictionary.

        Return the value for key if key is in the dictionary, else default.

    toJson()

    update([E], **F) → None. Update D from dict/iterable E and F.
        If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a
        .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

    values() → an object providing a view on D's values

    white_and_black_listed = 3

    white_listed = 1

class bitsharesbase.operations.Assert (*args, **kwargs)
    Bases: graphenebase.objects.GrapheneObject

    clear() → None. Remove all items from od.

    copy() → a shallow copy of od

    data
        Read data explicitly (backwards compatibility)
```

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E], **F$ ) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Asset\_claim\_fees** (\*args, \*\*kwargs)  
Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem()**

Remove and return a (key, value) pair from the dictionary.

Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**toJson()**

**update** ([*E*], \*\**F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** bitsharesbase.operations.**Asset\_claim\_pool** (\*args, \*\*kwargs)

Bases: graphenebase.objects.GrapheneObject

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**

Read data explicitly (backwards compatibility)

**fromkeys()**

Create a new ordered dictionary with keys from iterable and values set to value.

**get()**

Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**

Move an existing element to the end (or beginning if last is false).

Raise KeyError if the element does not exist.

**pop** (*k*, [*d*]) → v, remove specified key and return the corresponding

value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem()**

Remove and return a (key, value) pair from the dictionary.

Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**toJson()**

**update** ([*E*], \*\**F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values



```

class bitsharesbase.operations.Asset_create(*args, **kwargs)
    Bases: graphenebase.objects.GrapheneObject

    clear() → None. Remove all items from od.

    copy() → a shallow copy of od

    data
        Read data explicitly (backwards compatibility)

    fromkeys()
        Create a new ordered dictionary with keys from iterable and values set to value.

    get()
        Return the value for key if key is in the dictionary, else default.

    items() → a set-like object providing a view on D's items

    json()

    keys() → a set-like object providing a view on D's keys

    move_to_end()
        Move an existing element to the end (or beginning if last is false).

        Raise KeyError if the element does not exist.

    pop(k[, d]) → v, remove specified key and return the corresponding
        value. If key is not found, d is returned if given, otherwise KeyError is raised.

    popitem()
        Remove and return a (key, value) pair from the dictionary.

        Pairs are returned in LIFO order if last is true or FIFO order if false.

    setdefault()
        Insert key with a value of default if key is not in the dictionary.

        Return the value for key if key is in the dictionary, else default.

    toJson()

    update([E], **F) → None. Update D from dict/iterable E and F.
        If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a
        .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

    values() → an object providing a view on D's values

class bitsharesbase.operations.Asset_fund_fee_pool(*args, **kwargs)
    Bases: graphenebase.objects.GrapheneObject

    clear() → None. Remove all items from od.

    copy() → a shallow copy of od

    data
        Read data explicitly (backwards compatibility)

    fromkeys()
        Create a new ordered dictionary with keys from iterable and values set to value.

    get()
        Return the value for key if key is in the dictionary, else default.

    items() → a set-like object providing a view on D's items

    json()

```

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E], **F$ ) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Asset\_global\_settle** (\*args, \*\*kwargs)  
Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

```

toJson()
update([E], **F) → None. Update D from dict/iterable E and F.
    If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a
    .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values() → an object providing a view on D's values

class bitsharesbase.operations.Asset_issue(*args, **kwargs)
    Bases: graphenebase.objects.GrapheneObject
clear() → None. Remove all items from od.
copy() → a shallow copy of od
data
    Read data explicitly (backwards compatibility)
fromkeys()
    Create a new ordered dictionary with keys from iterable and values set to value.
get()
    Return the value for key if key is in the dictionary, else default.
items() → a set-like object providing a view on D's items
json()
keys() → a set-like object providing a view on D's keys
move_to_end()
    Move an existing element to the end (or beginning if last is false).
    Raise KeyError if the element does not exist.
pop(k, d) → v, remove specified key and return the corresponding
    value. If key is not found, d is returned if given, otherwise KeyError is raised.
popitem()
    Remove and return a (key, value) pair from the dictionary.
    Pairs are returned in LIFO order if last is true or FIFO order if false.
setdefault()
    Insert key with a value of default if key is not in the dictionary.
    Return the value for key if key is in the dictionary, else default.
toJson()
update([E], **F) → None. Update D from dict/iterable E and F.
    If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a
    .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values() → an object providing a view on D's values

class bitsharesbase.operations.Asset_publish_feed(*args, **kwargs)
    Bases: graphenebase.objects.GrapheneObject
clear() → None. Remove all items from od.
copy() → a shallow copy of od
data
    Read data explicitly (backwards compatibility)

```

**fromkeys** ()

Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()

Move an existing element to the end (or beginning if last is false).

Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()

Remove and return a (key, value) pair from the dictionary.

Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E], **F$ ) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Asset\_reserve** (\*args, \*\*kwargs)

Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**

Read data explicitly (backwards compatibility)

**fromkeys** ()

Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()

Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()

Move an existing element to the end (or beginning if last is false).

Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update([E], \*\*F) → None.** Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** bitsharesbase.operations.**Asset\_settle**(\*args, \*\*kwargs)  
Bases: graphenebase.objects.GrapheneObject

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**detail(\*args, \*\*kwargs)**

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(k[, d]) → v,** remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update([E], \*\*F) → None.** Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

```
class bitsharesbase.operations.Asset_update (*args, **kwargs)
    Bases: graphenebase.objects.GrapheneObject

    clear () → None. Remove all items from od.

    copy () → a shallow copy of od

    data
        Read data explicitly (backwards compatibility)

    fromkeys ()
        Create a new ordered dictionary with keys from iterable and values set to value.

    get ()
        Return the value for key if key is in the dictionary, else default.

    items () → a set-like object providing a view on D's items

    json ()

    keys () → a set-like object providing a view on D's keys

    move_to_end ()
        Move an existing element to the end (or beginning if last is false).

        Raise KeyError if the element does not exist.

    pop (k[, d]) → v, remove specified key and return the corresponding
        value. If key is not found, d is returned if given, otherwise KeyError is raised.

    popitem ()
        Remove and return a (key, value) pair from the dictionary.

        Pairs are returned in LIFO order if last is true or FIFO order if false.

    setdefault ()
        Insert key with a value of default if key is not in the dictionary.

        Return the value for key if key is in the dictionary, else default.

    toJson ()

    update ([E], **F) → None. Update D from dict/iterable E and F.
        If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a
        .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

    values () → an object providing a view on D's values

class bitsharesbase.operations.Asset_update_bitasset (*args, **kwargs)
    Bases: graphenebase.objects.GrapheneObject

    clear () → None. Remove all items from od.

    copy () → a shallow copy of od

    data
        Read data explicitly (backwards compatibility)

    fromkeys ()
        Create a new ordered dictionary with keys from iterable and values set to value.

    get ()
        Return the value for key if key is in the dictionary, else default.

    items () → a set-like object providing a view on D's items

    json ()
```

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
 Move an existing element to the end (or beginning if last is false).  
 Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
 Remove and return a (key, value) pair from the dictionary.  
 Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
 Insert key with a value of default if key is not in the dictionary.  
 Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E], **F$ ) → None. Update D from dict/iterable E and F.  
 If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Asset\_update\_feed\_producers** (\*args, \*\*kwargs)  
 Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
 Read data explicitly (backwards compatibility)

**fromkeys** ()  
 Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
 Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
 Move an existing element to the end (or beginning if last is false).  
 Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
 Remove and return a (key, value) pair from the dictionary.  
 Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
 Insert key with a value of default if key is not in the dictionary.  
 Return the value for key if key is in the dictionary, else default.

**toJson()**

**update**(*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.  
If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** `bitsharesbase.operations.Asset_update_issuer`(*\*args*, *\*\*kwargs*)  
Bases: `graphenebase.objects.GrapheneObject`

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise `KeyError` if the element does not exist.

**pop**(*k*, *d*) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise `KeyError` is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update**(*[E]*, *\*\*F*) → None. Update D from dict/iterable E and F.  
If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** `bitsharesbase.operations.Balance_claim`(*\*args*, *\*\*kwargs*)  
Bases: `graphenebase.objects.GrapheneObject`

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**detail**(*\*args*, *\*\*kwargs*)



**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ ) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E], **F$ ) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Bid\_collateral** (\*args, \*\*kwargs)  
Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**detail** (\*args, \*\*kwargs)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** ( $k[d]$ )  $\rightarrow v$ , remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise `KeyError` is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E]$ ,  $**F$ )  $\rightarrow$  None. Update D from dict/iterable E and F.  
If E is present and has a `.keys()` method, then does: for k in E:  $D[k] = E[k]$  If E is present and lacks a `.keys()` method, then does: for k, v in E:  $D[k] = v$  In either case, this is followed by: for k in F:  $D[k] = F[k]$

**values** ()  $\rightarrow$  an object providing a view on D's values

**class** `bitsharesbase.operations.Call_order_update` (\*args, \*\*kwargs)  
Bases: `graphenebase.objects.GrapheneObject`

**clear** ()  $\rightarrow$  None. Remove all items from od.

**copy** ()  $\rightarrow$  a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** ()  $\rightarrow$  a set-like object providing a view on D's items

**json** ()

**keys** ()  $\rightarrow$  a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise `KeyError` if the element does not exist.

**pop** ( $k[d]$ )  $\rightarrow v$ , remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise `KeyError` is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E]$ ,  $**F$ )  $\rightarrow$  None. Update D from dict/iterable E and F.  
If E is present and has a `.keys()` method, then does: for k in E:  $D[k] = E[k]$  If E is present and lacks a `.keys()` method, then does: for k, v in E:  $D[k] = v$  In either case, this is followed by: for k in F:  $D[k] = F[k]$

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Committee\_member\_create** (\*args, \*\*kwargs)  
 Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
 Read data explicitly (backwards compatibility)

**fromkeys** ()  
 Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
 Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
 Move an existing element to the end (or beginning if last is false).  
 Raise KeyError if the element does not exist.

**pop** (k[, d]) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
 Remove and return a (key, value) pair from the dictionary.  
 Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
 Insert key with a value of default if key is not in the dictionary.  
 Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ([E], \*\*F) → None. Update D from dict/iterable E and F.  
 If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Custom** (\*args, \*\*kwargs)  
 Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
 Read data explicitly (backwards compatibility)

**fromkeys** ()  
 Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
 Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(*k*[, *d*])** → *v*, remove specified key and return the corresponding value. If key is not found, *d* is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update([*E*], \*\**F*)** → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** bitsharesbase.operations.HtlcHash(*o*)  
Bases: graphenebase.types.Static\_variant

**elements** = [<class 'graphenebase.types.Ripemd160'>, <class 'graphenebase.types.Sha1'>]

**class** bitsharesbase.operations.Htlc\_create(\**args*, \*\**kwargs*)  
Bases: graphenebase.objects.GrapheneObject

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**detail(\**args*, \*\**kwargs*)**

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(*k*[, *d*])** → *v*, remove specified key and return the corresponding value. If key is not found, *d* is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update([E], \*\*F)** → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** bitsharesbase.operations.**Htlc\_extend**(\*args, \*\*kwargs)  
Bases: graphenebase.objects.GrapheneObject

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**detail(\*args, \*\*kwargs)**

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(k[, d])** → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update([E], \*\*F)** → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

```
class bitsharesbase.operations.Htlc_redeem (*args, **kwargs)
    Bases: graphenebase.objects.GrapheneObject

clear () → None. Remove all items from od.

copy () → a shallow copy of od

data
    Read data explicitly (backwards compatibility)

detail (*args, **kwargs)

fromkeys ()
    Create a new ordered dictionary with keys from iterable and values set to value.

get ()
    Return the value for key if key is in the dictionary, else default.

items () → a set-like object providing a view on D's items

json ()

keys () → a set-like object providing a view on D's keys

move_to_end ()
    Move an existing element to the end (or beginning if last is false).

    Raise KeyError if the element does not exist.

pop (k[, d]) → v, remove specified key and return the corresponding
    value. If key is not found, d is returned if given, otherwise KeyError is raised.

popitem ()
    Remove and return a (key, value) pair from the dictionary.

    Pairs are returned in LIFO order if last is true or FIFO order if false.

setdefault ()
    Insert key with a value of default if key is not in the dictionary.

    Return the value for key if key is in the dictionary, else default.

toJson ()

update ([E], **F) → None. Update D from dict/iterable E and F.
    If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a
    .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values () → an object providing a view on D's values

class bitsharesbase.operations.Limit_order_cancel (*args, **kwargs)
    Bases: graphenebase.objects.GrapheneObject

clear () → None. Remove all items from od.

copy () → a shallow copy of od

data
    Read data explicitly (backwards compatibility)

fromkeys ()
    Create a new ordered dictionary with keys from iterable and values set to value.

get ()
    Return the value for key if key is in the dictionary, else default.

items () → a set-like object providing a view on D's items
```

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(*k*[, *d*])** → *v*, remove specified key and return the corresponding value. If key is not found, *d* is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update([*E*], \*\**F*)** → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** bitsharesbase.operations.**Limit\_order\_create**(*\*args*, \*\**kwargs*)  
Bases: graphenebase.objects.GrapheneObject

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(*k*[, *d*])** → *v*, remove specified key and return the corresponding value. If key is not found, *d* is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Liquidity\_pool\_create** (\*args, *\*\*kwargs*)  
Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** (*k*, *d*) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Liquidity\_pool\_delete** (\*args, *\*\*kwargs*)  
Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od



**data**  
Read data explicitly (backwards compatibility)

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(*k*, *d*)** → *v*, remove specified key and return the corresponding value. If key is not found, *d* is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update([*E*], \*\**F*)** → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** bitsharesbase.operations.**Liquidity\_pool\_deposit** (\*args, \*\*kwargs)  
Bases: graphenebase.objects.GrapheneObject

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** ( $k[d]$ )  $\rightarrow v$ , remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise `KeyError` is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E]$ ,  $**F$ )  $\rightarrow$  None. Update D from dict/iterable E and F.  
If E is present and has a `.keys()` method, then does: for k in E:  $D[k] = E[k]$  If E is present and lacks a `.keys()` method, then does: for k, v in E:  $D[k] = v$  In either case, this is followed by: for k in F:  $D[k] = F[k]$

**values** ()  $\rightarrow$  an object providing a view on D's values

**class** bitsharesbase.operations.**Liquidity\_pool\_exchange** (\*args, \*\*kwargs)  
Bases: graphenebase.objects.GrapheneObject

**clear** ()  $\rightarrow$  None. Remove all items from od.

**copy** ()  $\rightarrow$  a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** ()  $\rightarrow$  a set-like object providing a view on D's items

**json** ()

**keys** ()  $\rightarrow$  a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise `KeyError` if the element does not exist.

**pop** ( $k[d]$ )  $\rightarrow v$ , remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise `KeyError` is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E]$ ,  $**F$ )  $\rightarrow$  None. Update D from dict/iterable E and F.  
If E is present and has a `.keys()` method, then does: for k in E:  $D[k] = E[k]$  If E is present and lacks a `.keys()` method, then does: for k, v in E:  $D[k] = v$  In either case, this is followed by: for k in F:  $D[k] = F[k]$

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Liquidity\_pool\_withdraw** (\*args, \*\*kwargs)  
 Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
 Read data explicitly (backwards compatibility)

**fromkeys** ()  
 Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
 Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
 Move an existing element to the end (or beginning if last is false).  
 Raise KeyError if the element does not exist.

**pop** (k[, d]) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
 Remove and return a (key, value) pair from the dictionary.  
 Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
 Insert key with a value of default if key is not in the dictionary.  
 Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ([E], \*\*F) → None. Update D from dict/iterable E and F.  
 If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Op\_wrapper** (\*args, \*\*kwargs)  
 Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
 Read data explicitly (backwards compatibility)

**fromkeys** ()  
 Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
 Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(*k*[, *d*])** → *v*, remove specified key and return the corresponding value. If key is not found, *d* is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update([*E*], \*\**F*)** → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** bitsharesbase.operations.**Override\_transfer**(*\*args*, *\*\*kwargs*)  
Bases: graphenebase.objects.GrapheneObject

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(*k*[, *d*])** → *v*, remove specified key and return the corresponding value. If key is not found, *d* is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
 Insert key with a value of default if key is not in the dictionary.  
 Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.  
 If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Proposal\_create** (\*args, *\*\*kwargs*)  
 Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
 Read data explicitly (backwards compatibility)

**fromkeys** ()  
 Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
 Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
 Move an existing element to the end (or beginning if last is false).  
 Raise KeyError if the element does not exist.

**pop** (*k*, [*d*]) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
 Remove and return a (key, value) pair from the dictionary.  
 Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
 Insert key with a value of default if key is not in the dictionary.  
 Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.  
 If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Proposal\_update** (\*args, *\*\*kwargs*)  
 Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(*k*, *d*)** → *v*, remove specified key and return the corresponding value. If key is not found, *d* is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update(*[E]*, *\*\*F*)** → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**class** bitsharesbase.operations.**Ticket\_create\_operation**(*\*args*, *\*\*kwargs*)  
Bases: graphenebase.objects.GrapheneObject

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** ( $k[, d]$ )  $\rightarrow v$ , remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise `KeyError` is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E]$ ,  $**F$ )  $\rightarrow$  None. Update D from dict/iterable E and F.  
If E is present and has a `.keys()` method, then does: for k in E:  $D[k] = E[k]$  If E is present and lacks a `.keys()` method, then does: for k, v in E:  $D[k] = v$  In either case, this is followed by: for k in F:  $D[k] = F[k]$

**values** ()  $\rightarrow$  an object providing a view on D's values

**class** `bitsharesbase.operations.Ticket_update_operation` (\*args, \*\*kwargs)  
Bases: `graphenebase.objects.GrapheneObject`

**clear** ()  $\rightarrow$  None. Remove all items from od.

**copy** ()  $\rightarrow$  a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** ()  $\rightarrow$  a set-like object providing a view on D's items

**json** ()

**keys** ()  $\rightarrow$  a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise `KeyError` if the element does not exist.

**pop** ( $k[, d]$ )  $\rightarrow v$ , remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise `KeyError` is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ( $[E]$ ,  $**F$ )  $\rightarrow$  None. Update D from dict/iterable E and F.  
If E is present and has a `.keys()` method, then does: for k in E:  $D[k] = E[k]$  If E is present and lacks a `.keys()` method, then does: for k, v in E:  $D[k] = v$  In either case, this is followed by: for k in F:  $D[k] = F[k]$

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Transfer** (\*args, \*\*kwargs)  
Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** (k[, d]) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ([E], \*\*F) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Vesting\_balance\_withdraw** (\*args, \*\*kwargs)  
Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items



**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(*k*, *d*)** → *v*, remove specified key and return the corresponding value. If key is not found, *d* is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update(*[E]*, *\*\*F*)** → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for *k* in E: D[*k*] = E[*k*] If E is present and lacks a .keys() method, then does: for *k*, *v* in E: D[*k*] = *v* In either case, this is followed by: for *k* in F: D[*k*] = F[*k*]

**values()** → an object providing a view on D's values

**class** bitsharesbase.operations.**Withdraw\_permission\_create**(*\*args*, *\*\*kwargs*)  
Bases: graphenebase.objects.GrapheneObject

**clear()** → None. Remove all items from od.

**copy()** → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(*k*, *d*)** → *v*, remove specified key and return the corresponding value. If key is not found, *d* is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Witness\_update** (\*args, *\*\*kwargs*)  
Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop** (*k*, *d*) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson** ()

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**class** bitsharesbase.operations.**Worker\_create** (\*args, *\*\*kwargs*)  
Bases: graphenebase.objects.GrapheneObject

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**fromkeys()**  
Create a new ordered dictionary with keys from iterable and values set to value.

**get()**  
Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**json()**

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**pop(*k*, *d*)** → *v*, remove specified key and return the corresponding value. If key is not found, *d* is returned if given, otherwise KeyError is raised.

**popitem()**  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**toJson()**

**update(*[E]*, *\*\*F*)** → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

`bitsharesbase.operations.fill_classmaps()`

`bitsharesbase.operations.getOperationClassForId(op_id)`  
Convert an operation id into the corresponding class.

`bitsharesbase.operations.getOperationIdForClass(name)`  
Convert an operation classname into the corresponding id.

`bitsharesbase.operations.getOperationNameForId(i)`  
Convert an operation id into the corresponding string.

## bitsharesbase.signedtransactions module

**class** `bitsharesbase.signedtransactions.Signed_Transaction(*args, **kwargs)`  
Bases: `graphenebase.signedtransactions.Signed_Transaction`  
Create a signed transaction and offer method to create the signature.

### Parameters

- **refNum** (*num*) – parameter `ref_block_num` (see `getBlockParams`)
- **refPrefix** (*num*) – parameter `ref_block_prefix` (see `getBlockParams`)

- **expiration** (*str*) – expiration date
- **operations** (*Array*) – array of operations

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**data**  
Read data explicitly (backwards compatibility)

**default\_prefix** = 'BTS'

**deriveDigest** (*chain*)

**detail** (\*args, \*\*kwargs)

**fromkeys** ()  
Create a new ordered dictionary with keys from iterable and values set to value.

**get** ()  
Return the value for key if key is in the dictionary, else default.

**getChainParams** (*chain*)

**getKnownChains** ()

**getOperationKlass** ()

**get\_default\_prefix** ()

**id**  
The transaction id of this transaction

**items** () → a set-like object providing a view on D's items

**json** ()

**keys** () → a set-like object providing a view on D's keys

**known\_chains** = {'BTS': {'chain\_id': '4018d7844c78f6a6c41c6a552b898022310fc5dec06da467'}}

**move\_to\_end** ()  
Move an existing element to the end (or beginning if last is false).  
Raise KeyError if the element does not exist.

**operation\_klass**  
alias of *bitsharesbase.objects.Operation*

**pop** (*k*, [*d*]) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem** ()  
Remove and return a (key, value) pair from the dictionary.  
Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault** ()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

**sign** (*wifkeys*, *chain=None*)  
Sign the transaction with the provided private keys.

#### Parameters

- **wifkeys** (*array*) – Array of wif keys
- **chain** (*str*) – identifier for the chain

**toJson** ()

**update** ([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values** () → an object providing a view on D's values

**verify** (*pubkeys*=[], *chain=None*)

## bitsharesbase.transactions module

### Module contents



### 6.1 Market Pegged Assets

Market Pegged Assets are a class of financial instruments available on the BitShares network. They implement a loan where one party gets stability with respect to an underlying asset and the other gets leverage against another digital asset that serves as backing collateral for the first.





## CHAPTER 7

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### b

`bitshares`, 179  
`bitshares.account`, 97  
`bitshares.aio`, 97  
`bitshares.aio.account`, 19  
`bitshares.aio.amount`, 24  
`bitshares.aio.asset`, 26  
`bitshares.aio.bitshares`, 30  
`bitshares.aio.block`, 41  
`bitshares.aio.blockchain`, 45  
`bitshares.aio.blockchainobject`, 48  
`bitshares.aio.committee`, 51  
`bitshares.aio.dex`, 53  
`bitshares.aio.genesisbalance`, 55  
`bitshares.aio.htlc`, 59  
`bitshares.aio.instance`, 61  
`bitshares.aio.market`, 62  
`bitshares.aio.memo`, 67  
`bitshares.aio.message`, 68  
`bitshares.aio.price`, 69  
`bitshares.aio.proposal`, 78  
`bitshares.aio.transactionbuilder`, 82  
`bitshares.aio.vesting`, 86  
`bitshares.aio.wallet`, 88  
`bitshares.aio.witness`, 90  
`bitshares.aio.worker`, 93  
`bitshares.amount`, 102  
`bitshares.asset`, 104  
`bitshares.bitshares`, 109  
`bitshares.block`, 120  
`bitshares.blockchain`, 124  
`bitshares.blockchainobject`, 127  
`bitshares.committee`, 130  
`bitshares.dex`, 132  
`bitshares.exceptions`, 134  
`bitshares.genesisbalance`, 135  
`bitshares.htlc`, 138  
`bitshares.instance`, 140  
`bitshares.market`, 141  
`bitshares.memo`, 146  
`bitshares.message`, 148  
`bitshares.notify`, 149  
`bitshares.price`, 151  
`bitshares.proposal`, 160  
`bitshares.storage`, 163  
`bitshares.transactionbuilder`, 164  
`bitshares.utils`, 167  
`bitshares.vesting`, 167  
`bitshares.wallet`, 169  
`bitshares.witness`, 171  
`bitshares.worker`, 175  
`bitsharesbase`, 225  
`bitsharesbase.account`, 179  
`bitsharesbase.asset_permissions`, 182  
`bitsharesbase.bip38`, 182  
`bitsharesbase.chains`, 183  
`bitsharesbase.memo`, 183  
`bitsharesbase.objects`, 183  
`bitsharesbase.objecttypes`, 191  
`bitsharesbase.operationids`, 191  
`bitsharesbase.operations`, 191  
`bitsharesbase.signedtransactions`, 223  
`bitsharesbase.transactions`, 225



## A

- account (*bitshares.account.AccountUpdate* attribute), 101
- account (*bitshares.aio.account.AccountUpdate* attribute), 22
- account (*bitshares.aio.committee.Committee* attribute), 51
- account (*bitshares.aio.vesting.Vesting* attribute), 86
- account (*bitshares.aio.witness.Witness* attribute), 90
- account (*bitshares.aio.worker.Worker* attribute), 94
- account (*bitshares.committee.Committee* attribute), 130
- account (*bitshares.vesting.Vesting* attribute), 167
- account (*bitshares.witness.Witness* attribute), 171
- account (*bitshares.worker.Worker* attribute), 175
- Account (class in *bitshares.account*), 97
- Account (class in *bitshares.aio.account*), 19
- Account\_create (class in *bitsharesbase.operations*), 191
- account\_id (*bitshares.aio.committee.Committee* attribute), 51
- account\_id (*bitshares.committee.Committee* attribute), 130
- Account\_update (class in *bitsharesbase.operations*), 192
- Account\_upgrade (class in *bitsharesbase.operations*), 193
- Account\_whitelist (class in *bitsharesbase.operations*), 193
- account\_whitelist () (*bitshares.aio.bitshares.BitShares* method), 30
- account\_whitelist () (*bitshares.bitshares.BitShares* method), 110
- AccountCreateExtensions (class in *bitsharesbase.objects*), 183
- AccountCreateExtensions.Buyback\_options (class in *bitsharesbase.objects*), 183
- AccountCreateExtensions.Null\_ext (class in *bitsharesbase.objects*), 184
- AccountExistsException, 134
- AccountId () (in module *bitsharesbase.objects*), 184
- accountopenorders () (*bitshares.aio.market.Market* method), 62
- accountopenorders () (*bitshares.market.Market* method), 142
- AccountOptions (class in *bitsharesbase.objects*), 184
- accounttrades () (*bitshares.aio.market.Market* method), 62
- accounttrades () (*bitshares.market.Market* method), 142
- AccountUpdate (class in *bitshares.account*), 100
- AccountUpdate (class in *bitshares.aio.account*), 22
- add () (*bitsharesbase.account.PublicKey* method), 182
- add\_authorities () (*bitshares.aio.asset.Asset* method), 26
- add\_authorities () (*bitshares.asset.Asset* method), 104
- add\_markets () (*bitshares.aio.asset.Asset* method), 26
- add\_markets () (*bitshares.asset.Asset* method), 104
- add\_required\_fees () (*bitshares.aio.transactionbuilder.TransactionBuilder* method), 83
- add\_required\_fees () (*bitshares.transactionbuilder.TransactionBuilder* method), 165
- addPrivateKey () (*bitshares.aio.wallet.Wallet* method), 88
- addPrivateKey () (*bitshares.wallet.Wallet* method), 169
- address (*bitsharesbase.account.PrivateKey* attribute), 181
- address (*bitsharesbase.account.PublicKey* attribute), 182
- Address (class in *bitsharesbase.account*), 179
- addSigningInformation () (*bitshares.aio.transactionbuilder.TransactionBuilder* method), 83

- `addSigningInformation()` (*bitshares.transactionbuilder.TransactionBuilder method*), 165
- `adjust_collateral_ratio()` (*bitshares.aio.dex.Dex method*), 53
- `adjust_collateral_ratio()` (*bitshares.dex.Dex method*), 132
- `adjust_debt()` (*bitshares.aio.dex.Dex method*), 54
- `adjust_debt()` (*bitshares.dex.Dex method*), 133
- `allow()` (*bitshares.aio.bitshares.BitShares method*), 31
- `allow()` (*bitshares.bitshares.BitShares method*), 110
- `amount` (*bitshares.aio.amount.Amount attribute*), 24
- `amount` (*bitshares.amount.Amount attribute*), 103
- `Amount` (*class in bitshares.aio.amount*), 24
- `Amount` (*class in bitshares.amount*), 102
- `append()` (*bitshares.aio.genesisbalance.GenesisBalances method*), 58
- `append()` (*bitshares.aio.proposal.Proposals method*), 80
- `append()` (*bitshares.aio.witness.Witnesses method*), 92
- `append()` (*bitshares.aio.worker.Workers method*), 95
- `append()` (*bitshares.genesisbalance.GenesisBalances method*), 137
- `append()` (*bitshares.proposal.Proposals method*), 162
- `append()` (*bitshares.witness.Witnesses method*), 173
- `append()` (*bitshares.worker.Workers method*), 177
- `append()` (*bitsharesbase.objects.Operation method*), 188
- `appendMissingSignatures()` (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 84
- `appendMissingSignatures()` (*bitshares.transactionbuilder.TransactionBuilder method*), 165
- `appendOps()` (*bitshares.aio.transactionbuilder.ProposalBuilder method*), 82
- `appendOps()` (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 84
- `appendOps()` (*bitshares.transactionbuilder.ProposalBuilder method*), 164
- `appendOps()` (*bitshares.transactionbuilder.TransactionBuilder method*), 165
- `appendSigner()` (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 84
- `appendSigner()` (*bitshares.transactionbuilder.TransactionBuilder method*), 165
- `appendWif()` (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 84
- `appendWif()` (*bitshares.transactionbuilder.TransactionBuilder method*), 165
- `approvecommittee()` (*bitshares.aio.bitshares.BitShares method*), 31
- `approvecommittee()` (*bitshares.bitshares.BitShares method*), 110
- `approveproposal()` (*bitshares.aio.bitshares.BitShares method*), 31
- `approveproposal()` (*bitshares.bitshares.BitShares method*), 110
- `approvewitness()` (*bitshares.aio.bitshares.BitShares method*), 31
- `approvewitness()` (*bitshares.bitshares.BitShares method*), 111
- `approveworker()` (*bitshares.aio.bitshares.BitShares method*), 31
- `approveworker()` (*bitshares.bitshares.BitShares method*), 111
- `args` (*bitshares.exceptions.AccountExistsException attribute*), 134
- `args` (*bitshares.exceptions.HtlcDoesNotExistException attribute*), 135
- `args` (*bitshares.exceptions.ObjectNotInProposalBuffer attribute*), 135
- `args` (*bitshares.exceptions.RPCConnectionRequired attribute*), 135
- `as_base()` (*bitshares.aio.price.FilledOrder method*), 70
- `as_base()` (*bitshares.aio.price.Order method*), 71
- `as_base()` (*bitshares.aio.price.Price method*), 74
- `as_base()` (*bitshares.aio.price.UpdateCallOrder method*), 77
- `as_base()` (*bitshares.price.FilledOrder method*), 151
- `as_base()` (*bitshares.price.Order method*), 153
- `as_base()` (*bitshares.price.Price method*), 155
- `as_base()` (*bitshares.price.UpdateCallOrder method*), 158
- `as_quote()` (*bitshares.aio.price.FilledOrder method*), 70
- `as_quote()` (*bitshares.aio.price.Order method*), 71
- `as_quote()` (*bitshares.aio.price.Price method*), 74
- `as_quote()` (*bitshares.aio.price.UpdateCallOrder method*), 77
- `as_quote()` (*bitshares.price.FilledOrder method*), 151
- `as_quote()` (*bitshares.price.Order method*), 153
- `as_quote()` (*bitshares.price.Price method*), 155
- `as_quote()` (*bitshares.price.UpdateCallOrder method*), 158
- `Assert` (*class in bitsharesbase.operations*), 194
- `AssertPredicate` (*class in bitsharesbase.objects*), 185
- `asset` (*bitshares.aio.amount.Amount attribute*), 24
- `asset` (*bitshares.amount.Amount attribute*), 103
- `Asset` (*class in bitshares.aio.asset*), 26
- `Asset` (*class in bitshares.asset*), 104

- Asset\_claim\_fees (class in *bitsharesbase.operations*), 195  
 Asset\_claim\_pool (class in *bitsharesbase.operations*), 196  
 Asset\_create (class in *bitsharesbase.operations*), 196  
 Asset\_fund\_fee\_pool (class in *bitsharesbase.operations*), 197  
 Asset\_global\_settle (class in *bitsharesbase.operations*), 198  
 Asset\_issue (class in *bitsharesbase.operations*), 199  
 Asset\_publish\_feed (class in *bitsharesbase.operations*), 199  
 Asset\_reserve (class in *bitsharesbase.operations*), 200  
 Asset\_settle (class in *bitsharesbase.operations*), 201  
 asset\_settle() (*bitshares.aio.bitshares.BitShares* method), 31  
 asset\_settle() (*bitshares.bitshares.BitShares* method), 111  
 Asset\_update (class in *bitsharesbase.operations*), 201  
 Asset\_update\_bitasset (class in *bitsharesbase.operations*), 202  
 Asset\_update\_feed\_producers (class in *bitsharesbase.operations*), 203  
 Asset\_update\_issuer (class in *bitsharesbase.operations*), 204  
 AssetId() (in module *bitsharesbase.objects*), 185  
 AssetOptions (class in *bitsharesbase.objects*), 185  
 awaitTxConfirmation() (*bitshares.aio.blockchain.Blockchain* method), 45  
 awaitTxConfirmation() (*bitshares.blockchain.Blockchain* method), 124
- ## B
- balance() (*bitshares.account.Account* method), 98  
 balance() (*bitshares.aio.account.Account* method), 20  
 Balance\_claim (class in *bitsharesbase.operations*), 204  
 balances (*bitshares.account.Account* attribute), 98  
 balances (*bitshares.aio.account.Account* attribute), 20  
 Bid\_collateral (class in *bitsharesbase.operations*), 205  
 bid\_collateral() (*bitshares.aio.bitshares.BitShares* method), 31  
 bid\_collateral() (*bitshares.bitshares.BitShares* method), 111  
 BitAssetOptions (class in *bitsharesbase.objects*), 186  
 bitcoin (*bitsharesbase.account.PrivateKey* attribute), 181  
 bitshares (*bitshares.account.Account* attribute), 98  
 bitshares (*bitshares.account.AccountUpdate* attribute), 101  
 bitshares (*bitshares.aio.account.Account* attribute), 20  
 bitshares (*bitshares.aio.account.AccountUpdate* attribute), 22  
 bitshares (*bitshares.aio.amount.Amount* attribute), 24  
 bitshares (*bitshares.aio.asset.Asset* attribute), 26  
 bitshares (*bitshares.aio.block.Block* attribute), 41  
 bitshares (*bitshares.aio.block.BlockHeader* attribute), 43  
 bitshares (*bitshares.aio.blockchain.Blockchain* attribute), 45  
 bitshares (*bitshares.aio.blockchainobject.BlockchainObject* attribute), 48  
 bitshares (*bitshares.aio.blockchainobject.Object* attribute), 49  
 bitshares (*bitshares.aio.committee.Committee* attribute), 51  
 bitshares (*bitshares.aio.dex.Dex* attribute), 54  
 bitshares (*bitshares.aio.genesisbalance.GenesisBalance* attribute), 56  
 bitshares (*bitshares.aio.genesisbalance.GenesisBalances* attribute), 58  
 bitshares (*bitshares.aio.htlc.Htlc* attribute), 59  
 bitshares (*bitshares.aio.instance.BlockchainInstance* attribute), 61  
 bitshares (*bitshares.aio.market.Market* attribute), 63  
 bitshares (*bitshares.aio.memo.Memo* attribute), 67  
 bitshares (*bitshares.aio.message.Message* attribute), 68  
 bitshares (*bitshares.aio.price.FilledOrder* attribute), 70  
 bitshares (*bitshares.aio.price.Order* attribute), 71  
 bitshares (*bitshares.aio.price.Price* attribute), 74  
 bitshares (*bitshares.aio.price.PriceFeed* attribute), 75  
 bitshares (*bitshares.aio.price.UpdateCallOrder* attribute), 77  
 bitshares (*bitshares.aio.proposal.Proposal* attribute), 78  
 bitshares (*bitshares.aio.proposal.Proposals* attribute), 80  
 bitshares (*bitshares.aio.transactionbuilder.ProposalBuilder* attribute), 82  
 bitshares (*bitshares.aio.transactionbuilder.TransactionBuilder* attribute), 84  
 bitshares (*bitshares.aio.vesting.Vesting* attribute), 86  
 bitshares (*bitshares.aio.wallet.Wallet* attribute), 88  
 bitshares (*bitshares.aio.witness.Witness* attribute), 90

`bitshares` (*bitshares.aio.witness.Witnesses* attribute), 92

`bitshares` (*bitshares.aio.worker.Worker* attribute), 94

`bitshares` (*bitshares.aio.worker.Workers* attribute), 96

`bitshares` (*bitshares.amount.Amount* attribute), 103

`bitshares` (*bitshares.asset.Asset* attribute), 105

`bitshares` (*bitshares.block.Block* attribute), 120

`bitshares` (*bitshares.block.BlockHeader* attribute), 122

`bitshares` (*bitshares.blockchain.Blockchain* attribute), 124

`bitshares` (*bitshares.blockchainobject.BlockchainObject* attribute), 127

`bitshares` (*bitshares.blockchainobject.Object* attribute), 128

`bitshares` (*bitshares.committee.Committee* attribute), 130

`bitshares` (*bitshares.dex.Dex* attribute), 133

`bitshares` (*bitshares.genesisbalance.GenesisBalance* attribute), 135

`bitshares` (*bitshares.genesisbalance.GenesisBalances* attribute), 137

`bitshares` (*bitshares.htlc.Htlc* attribute), 139

`bitshares` (*bitshares.instance.BlockchainInstance* attribute), 140

`bitshares` (*bitshares.market.Market* attribute), 142

`bitshares` (*bitshares.memo.Memo* attribute), 147

`bitshares` (*bitshares.message.Message* attribute), 148

`bitshares` (*bitshares.notify.Notify* attribute), 150

`bitshares` (*bitshares.price.FilledOrder* attribute), 151

`bitshares` (*bitshares.price.Order* attribute), 153

`bitshares` (*bitshares.price.Price* attribute), 155

`bitshares` (*bitshares.price.PriceFeed* attribute), 157

`bitshares` (*bitshares.price.UpdateCallOrder* attribute), 158

`bitshares` (*bitshares.proposal.Proposal* attribute), 160

`bitshares` (*bitshares.proposal.Proposals* attribute), 162

`bitshares` (*bitshares.transactionbuilder.ProposalBuilder* attribute), 164

`bitshares` (*bitshares.transactionbuilder.TransactionBuilder* attribute), 165

`bitshares` (*bitshares.vesting.Vesting* attribute), 167

`bitshares` (*bitshares.wallet.Wallet* attribute), 169

`bitshares` (*bitshares.witness.Witness* attribute), 171

`bitshares` (*bitshares.witness.Witnesses* attribute), 173

`bitshares` (*bitshares.worker.Worker* attribute), 175

`bitshares` (*bitshares.worker.Workers* attribute), 177

`BitShares` (class in `bitshares.aio.bitshares`), 30

`BitShares` (class in `bitshares.bitshares`), 109

`bitshares` (module), 179

`bitshares.account` (module), 97

`bitshares.aio` (module), 97

`bitshares.aio.account` (module), 19

`bitshares.aio.amount` (module), 24

`bitshares.aio.asset` (module), 26

`bitshares.aio.bitshares` (module), 30

`bitshares.aio.block` (module), 41

`bitshares.aio.blockchain` (module), 45

`bitshares.aio.blockchainobject` (module), 48

`bitshares.aio.committee` (module), 51

`bitshares.aio.dex` (module), 53

`bitshares.aio.genesisbalance` (module), 55

`bitshares.aio.htlc` (module), 59

`bitshares.aio.instance` (module), 61

`bitshares.aio.market` (module), 62

`bitshares.aio.memo` (module), 67

`bitshares.aio.message` (module), 68

`bitshares.aio.price` (module), 69

`bitshares.aio.proposal` (module), 78

`bitshares.aio.transactionbuilder` (module), 82

`bitshares.aio.vesting` (module), 86

`bitshares.aio.wallet` (module), 88

`bitshares.aio.witness` (module), 90

`bitshares.aio.worker` (module), 93

`bitshares.amount` (module), 102

`bitshares.asset` (module), 104

`bitshares.bitshares` (module), 109

`bitshares.block` (module), 120

`bitshares.blockchain` (module), 124

`bitshares.blockchainobject` (module), 127

`bitshares.committee` (module), 130

`bitshares.dex` (module), 132

`bitshares.exceptions` (module), 134

`bitshares.genesisbalance` (module), 135

`bitshares.htlc` (module), 138

`bitshares.instance` (module), 140

`bitshares.market` (module), 141

`bitshares.memo` (module), 146

`bitshares.message` (module), 148

`bitshares.notify` (module), 149

`bitshares.price` (module), 151

`bitshares.proposal` (module), 160

`bitshares.storage` (module), 163

`bitshares.transactionbuilder` (module), 164

`bitshares.utils` (module), 167

`bitshares.vesting` (module), 167

`bitshares.wallet` (module), 169

`bitshares.witness` (module), 171

`bitshares.worker` (module), 175

`bitsharesbase` (module), 225

`bitsharesbase.account` (module), 179

`bitsharesbase.asset_permissions` (module), 182

`bitsharesbase.bip38` (module), 182



- `bitsharesbase.chains (module)`, 183
- `bitsharesbase.memo (module)`, 183
- `bitsharesbase.objects (module)`, 183
- `bitsharesbase.objecttypes (module)`, 191
- `bitsharesbase.operationids (module)`, 191
- `bitsharesbase.operations (module)`, 191
- `bitsharesbase.signedtransactions (module)`, 223
- `bitsharesbase.transactions (module)`, 225
- `black_listed (bitsharesbase.operations.Account_whitelist attribute)`, 194
- `blacklist () (bitshares.account.Account method)`, 98
- `blacklist () (bitshares.aio.account.Account method)`, 20
- `Block (class in bitshares.aio.block)`, 41
- `Block (class in bitshares.block)`, 120
- `block_time () (bitshares.aio.blockchain.Blockchain method)`, 45
- `block_time () (bitshares.blockchain.Blockchain method)`, 124
- `block_timestamp () (bitshares.aio.blockchain.Blockchain method)`, 45
- `block_timestamp () (bitshares.blockchain.Blockchain method)`, 124
- `blockchain (bitshares.account.Account attribute)`, 98
- `blockchain (bitshares.account.AccountUpdate attribute)`, 101
- `blockchain (bitshares.aio.account.Account attribute)`, 20
- `blockchain (bitshares.aio.account.AccountUpdate attribute)`, 23
- `blockchain (bitshares.aio.amount.Amount attribute)`, 24
- `blockchain (bitshares.aio.asset.Asset attribute)`, 26
- `blockchain (bitshares.aio.block.Block attribute)`, 41
- `blockchain (bitshares.aio.block.BlockHeader attribute)`, 43
- `blockchain (bitshares.aio.blockchain.Blockchain attribute)`, 45
- `blockchain (bitshares.aio.blockchainobject.BlockchainObject attribute)`, 48
- `blockchain (bitshares.aio.blockchainobject.Object attribute)`, 49
- `blockchain (bitshares.aio.committee.Committee attribute)`, 51
- `blockchain (bitshares.aio.dex.Dex attribute)`, 54
- `blockchain (bitshares.aio.genesisbalance.GenesisBalance attribute)`, 56
- `blockchain (bitshares.aio.genesisbalance.GenesisBalances attribute)`, 58
- `blockchain (bitshares.aio.htlc.Htlc attribute)`, 59
- `blockchain (bitshares.aio.instance.BlockchainInstance attribute)`, 61
- `blockchain (bitshares.aio.market.Market attribute)`, 63
- `blockchain (bitshares.aio.memo.Memo attribute)`, 67
- `blockchain (bitshares.aio.message.Message attribute)`, 68
- `blockchain (bitshares.aio.price.FilledOrder attribute)`, 70
- `blockchain (bitshares.aio.price.Order attribute)`, 71
- `blockchain (bitshares.aio.price.Price attribute)`, 74
- `blockchain (bitshares.aio.price.PriceFeed attribute)`, 76
- `blockchain (bitshares.aio.price.UpdateCallOrder attribute)`, 77
- `blockchain (bitshares.aio.proposal.Proposal attribute)`, 78
- `blockchain (bitshares.aio.proposal.Proposals attribute)`, 80
- `blockchain (bitshares.aio.transactionbuilder.ProposalBuilder attribute)`, 82
- `blockchain (bitshares.aio.transactionbuilder.TransactionBuilder attribute)`, 84
- `blockchain (bitshares.aio.vesting.Vesting attribute)`, 86
- `blockchain (bitshares.aio.wallet.Wallet attribute)`, 88
- `blockchain (bitshares.aio.witness.Witness attribute)`, 90
- `blockchain (bitshares.aio.witness.Witnesses attribute)`, 92
- `blockchain (bitshares.aio.worker.Worker attribute)`, 94
- `blockchain (bitshares.aio.worker.Workers attribute)`, 96
- `blockchain (bitshares.amount.Amount attribute)`, 103
- `blockchain (bitshares.asset.Asset attribute)`, 105
- `blockchain (bitshares.block.Block attribute)`, 120
- `blockchain (bitshares.block.BlockHeader attribute)`, 122
- `blockchain (bitshares.blockchain.Blockchain attribute)`, 124
- `blockchain (bitshares.blockchainobject.BlockchainObject attribute)`, 127
- `blockchain (bitshares.blockchainobject.Object attribute)`, 128
- `blockchain (bitshares.committee.Committee attribute)`, 130
- `blockchain (bitshares.dex.Dex attribute)`, 133
- `blockchain (bitshares.genesisbalance.GenesisBalance attribute)`, 135
- `blockchain (bitshares.genesisbalance.GenesisBalances attribute)`, 137
- `blockchain (bitshares.htlc.Htlc attribute)`, 139
- `blockchain (bitshares.instance.BlockchainInstance attribute)`, 140

- blockchain (*bitshares.market.Market* attribute), 142
- blockchain (*bitshares.memo.Memo* attribute), 147
- blockchain (*bitshares.message.Message* attribute), 148
- blockchain (*bitshares.notify.Notify* attribute), 150
- blockchain (*bitshares.price.FilledOrder* attribute), 151
- blockchain (*bitshares.price.Order* attribute), 153
- blockchain (*bitshares.price.Price* attribute), 155
- blockchain (*bitshares.price.PriceFeed* attribute), 157
- blockchain (*bitshares.price.UpdateCallOrder* attribute), 158
- blockchain (*bitshares.proposal.Proposal* attribute), 160
- blockchain (*bitshares.proposal.Proposals* attribute), 162
- blockchain (*bitshares.transactionbuilder.ProposalBuilder* attribute), 164
- blockchain (*bitshares.transactionbuilder.TransactionBuilder* attribute), 165
- blockchain (*bitshares.vesting.Vesting* attribute), 167
- blockchain (*bitshares.wallet.Wallet* attribute), 169
- blockchain (*bitshares.witness.Witness* attribute), 171
- blockchain (*bitshares.witness.Witnesses* attribute), 173
- blockchain (*bitshares.worker.Worker* attribute), 175
- blockchain (*bitshares.worker.Workers* attribute), 177
- Blockchain (class in *bitshares.aio.blockchain*), 45
- Blockchain (class in *bitshares.blockchain*), 124
- blockchain\_instance\_class (*bitshares.account.Account* attribute), 98
- blockchain\_instance\_class (*bitshares.account.AccountUpdate* attribute), 101
- blockchain\_instance\_class (*bitshares.aio.account.Account* attribute), 20
- blockchain\_instance\_class (*bitshares.aio.account.AccountUpdate* attribute), 23
- blockchain\_instance\_class (*bitshares.aio.amount.Amount* attribute), 24
- blockchain\_instance\_class (*bitshares.aio.asset.Asset* attribute), 26
- blockchain\_instance\_class (*bitshares.aio.block.Block* attribute), 41
- blockchain\_instance\_class (*bitshares.aio.block.BlockHeader* attribute), 43
- blockchain\_instance\_class (*bitshares.aio.blockchain.Blockchain* attribute), 45
- blockchain\_instance\_class (*bitshares.aio.blockchainobject.BlockchainObject* attribute), 48
- blockchain\_instance\_class (*bitshares.aio.blockchainobject.Object* attribute), 49
- blockchain\_instance\_class (*bitshares.aio.committee.Committee* attribute), 52
- blockchain\_instance\_class (*bitshares.aio.genesisbalance.GenesisBalance* attribute), 56
- blockchain\_instance\_class (*bitshares.aio.genesisbalance.GenesisBalances* attribute), 58
- blockchain\_instance\_class (*bitshares.aio.htlc.Htlc* attribute), 59
- blockchain\_instance\_class (*bitshares.aio.market.Market* attribute), 63
- blockchain\_instance\_class (*bitshares.aio.memo.Memo* attribute), 67
- blockchain\_instance\_class (*bitshares.aio.message.Message* attribute), 68
- blockchain\_instance\_class (*bitshares.aio.price.FilledOrder* attribute), 70
- blockchain\_instance\_class (*bitshares.aio.price.Order* attribute), 71
- blockchain\_instance\_class (*bitshares.aio.price.Price* attribute), 74
- blockchain\_instance\_class (*bitshares.aio.price.PriceFeed* attribute), 76
- blockchain\_instance\_class (*bitshares.aio.price.UpdateCallOrder* attribute), 77
- blockchain\_instance\_class (*bitshares.aio.proposal.Proposal* attribute), 78
- blockchain\_instance\_class (*bitshares.aio.proposal.Proposals* attribute), 80
- blockchain\_instance\_class (*bitshares.aio.transactionbuilder.ProposalBuilder* attribute), 82
- blockchain\_instance\_class (*bitshares.aio.transactionbuilder.TransactionBuilder* attribute), 84
- blockchain\_instance\_class (*bitshares.aio.vesting.Vesting* attribute), 86
- blockchain\_instance\_class (*bitshares.aio.wallet.Wallet* attribute), 88
- blockchain\_instance\_class (*bitshares.aio.witness.Witness* attribute), 90
- blockchain\_instance\_class (*bitshares.aio.witness.Witnesses* attribute), 92
- blockchain\_instance\_class (*bitshares.aio.worker.Worker* attribute), 94
- blockchain\_instance\_class

- shares.aio.worker.Workers attribute*), 96
  - blockchain\_instance\_class (*bit-*  
*shares.amount.Amount attribute*), 103
  - blockchain\_instance\_class (*bit-*  
*shares.asset.Asset attribute*), 105
  - blockchain\_instance\_class (*bit-*  
*shares.block.Block attribute*), 120
  - blockchain\_instance\_class (*bit-*  
*shares.block.BlockHeader attribute*), 122
  - blockchain\_instance\_class (*bit-*  
*shares.blockchain.Blockchain attribute*), 124
  - blockchain\_instance\_class (*bit-*  
*shares.blockchainobject.BlockchainObject attribute*), 127
  - blockchain\_instance\_class (*bit-*  
*shares.blockchainobject.Object attribute*), 128
  - blockchain\_instance\_class (*bit-*  
*shares.committee.Committee attribute*), 130
  - blockchain\_instance\_class (*bit-*  
*shares.genesisbalance.GenesisBalance attribute*), 135
  - blockchain\_instance\_class (*bit-*  
*shares.genesisbalance.GenesisBalances attribute*), 137
  - blockchain\_instance\_class (*bit-*  
*shares.htlc.Htlc attribute*), 139
  - blockchain\_instance\_class (*bit-*  
*shares.market.Market attribute*), 142
  - blockchain\_instance\_class (*bit-*  
*shares.memo.Memo attribute*), 147
  - blockchain\_instance\_class (*bit-*  
*shares.message.Message attribute*), 148
  - blockchain\_instance\_class (*bit-*  
*shares.price.FilledOrder attribute*), 151
  - blockchain\_instance\_class (*bit-*  
*shares.price.Order attribute*), 153
  - blockchain\_instance\_class (*bit-*  
*shares.price.Price attribute*), 155
  - blockchain\_instance\_class (*bit-*  
*shares.price.PriceFeed attribute*), 157
  - blockchain\_instance\_class (*bit-*  
*shares.price.UpdateCallOrder attribute*), 158
  - blockchain\_instance\_class (*bit-*  
*shares.proposal.Proposal attribute*), 160
  - blockchain\_instance\_class (*bit-*  
*shares.proposal.Proposals attribute*), 162
  - blockchain\_instance\_class (*bit-*  
*shares.transactionbuilder.ProposalBuilder attribute*), 164
  - blockchain\_instance\_class (*bit-*  
*shares.transactionbuilder.TransactionBuilder attribute*), 166
  - blockchain\_instance\_class (*bit-*  
*shares.vesting.Vesting attribute*), 167
  - blockchain\_instance\_class (*bit-*  
*shares.wallet.Wallet attribute*), 169
  - blockchain\_instance\_class (*bit-*  
*shares.witness.Witness attribute*), 171
  - blockchain\_instance\_class (*bit-*  
*shares.witness.Witnesses attribute*), 173
  - blockchain\_instance\_class (*bit-*  
*shares.worker.Worker attribute*), 175
  - blockchain\_instance\_class (*bit-*  
*shares.worker.Workers attribute*), 177
  - BlockchainInstance (class in *bit-*  
*shares.aio.instance*), 61
  - BlockchainInstance (class in *bit-*  
*shares.aio.blockchainobject*), 48
  - BlockchainObject (class in *bit-*  
*shares.aio.blockchainobject*), 127
  - BlockchainObject (class in *bit-*  
*shares.aio.blockchainobject*), 127
  - BlockHeader (class in *bit-*  
*shares.aio.blockchainobject*), 43
  - BlockHeader (class in *bit-*  
*shares.aio.blockchainobject*), 122
  - blocks () (*bit-*  
*shares.aio.blockchainobject*), 45
  - blocks () (*bit-*  
*shares.aio.blockchainobject*), 124
  - borrow () (*bit-*  
*shares.aio.dex.Dex method*), 54
  - borrow () (*bit-*  
*shares.dex.Dex method*), 133
  - BrainKey (class in *bit-*  
*sharesbase.account*), 179
  - broadcast () (*bit-*  
*shares.aio.bitshares.BitShares method*), 31
  - broadcast () (*bit-*  
*shares.aio.transactionbuilder.ProposalBuilder method*), 83
  - broadcast () (*bit-*  
*shares.aio.transactionbuilder.TransactionBuilder method*), 84
  - broadcast () (*bit-*  
*shares.bitshares.BitShares method*), 111
  - broadcast () (*bit-*  
*shares.transactionbuilder.ProposalBuilder method*), 164
  - broadcast () (*bit-*  
*shares.transactionbuilder.TransactionBuilder method*), 166
  - buy () (*bit-*  
*shares.aio.market.Market method*), 63
  - buy () (*bit-*  
*shares.market.Market method*), 142
- ## C
- cache () (*bit-*  
*shares.aio.proposal.Proposals method*), 80
  - cache () (*bit-*  
*shares.aio.witness.Witnesses method*), 92
  - cache () (*bit-*  
*shares.aio.worker.Workers method*), 96
  - cache () (*bit-*  
*shares.proposal.Proposals method*), 162
  - cache () (*bit-*  
*shares.witness.Witnesses method*), 173
  - cache () (*bit-*  
*shares.worker.Workers method*), 177
  - cache\_object () (*bit-*  
*shares.account.Account class method*), 98

`cache_object()` (*bitshares.aio.account.Account* class method), 20  
`cache_object()` (*bitshares.aio.asset.Asset* class method), 26  
`cache_object()` (*bitshares.aio.block.Block* class method), 41  
`cache_object()` (*bitshares.aio.block.BlockHeader* class method), 43  
`cache_object()` (*bitshares.aio.blockchainobject.BlockchainObject* class method), 48  
`cache_object()` (*bitshares.aio.blockchainobject.Object* class method), 49  
`cache_object()` (*bitshares.aio.committee.Committee* class method), 52  
`cache_object()` (*bitshares.aio.genesisbalance.GenesisBalance* class method), 56  
`cache_object()` (*bitshares.aio.htlc.Htlc* class method), 59  
`cache_object()` (*bitshares.aio.proposal.Proposal* class method), 78  
`cache_object()` (*bitshares.aio.vesting.Vesting* class method), 86  
`cache_object()` (*bitshares.aio.witness.Witness* class method), 90  
`cache_object()` (*bitshares.aio.worker.Worker* class method), 94  
`cache_object()` (*bitshares.asset.Asset* class method), 105  
`cache_object()` (*bitshares.block.Block* class method), 120  
`cache_object()` (*bitshares.block.BlockHeader* class method), 122  
`cache_object()` (*bitshares.blockchainobject.BlockchainObject* class method), 127  
`cache_object()` (*bitshares.blockchainobject.Object* class method), 128  
`cache_object()` (*bitshares.committee.Committee* class method), 131  
`cache_object()` (*bitshares.genesisbalance.GenesisBalance* class method), 135  
`cache_object()` (*bitshares.htlc.Htlc* class method), 139  
`cache_object()` (*bitshares.proposal.Proposal* class method), 160  
`cache_object()` (*bitshares.vesting.Vesting* class method), 168  
`cache_object()` (*bitshares.witness.Witness* class method), 171  
`cache_object()` (*bitshares.worker.Worker* class method), 175  
`cache_objects()` (*bitshares.aio.proposal.Proposals* class method), 81  
`cache_objects()` (*bitshares.aio.witness.Witnesses* class method), 92  
`cache_objects()` (*bitshares.aio.worker.Workers* class method), 96  
`cache_objects()` (*bitshares.proposal.Proposals* class method), 162  
`cache_objects()` (*bitshares.witness.Witnesses* class method), 174  
`cache_objects()` (*bitshares.worker.Workers* class method), 177  
`Call_order_update` (class in *bitshares-base.operations*), 206  
`call_positions` (*bitshares.account.Account* attribute), 98  
`call_positions` (*bitshares.aio.account.Account* attribute), 20  
`CallOrderExtension` (class in *bitshares-base.objects*), 187  
`callpositions` (*bitshares.account.Account* attribute), 98  
`callpositions` (*bitshares.aio.account.Account* attribute), 20  
`calls` (*bitshares.aio.asset.Asset* attribute), 26  
`calls` (*bitshares.asset.Asset* attribute), 105  
`cancel()` (*bitshares.aio.bitshares.BitShares* method), 32  
`cancel()` (*bitshares.aio.market.Market* method), 63  
`cancel()` (*bitshares.bitshares.BitShares* method), 111  
`cancel()` (*bitshares.market.Market* method), 143  
`cancel_subscriptions()` (*bitshares.aio.bitshares.BitShares* method), 32  
`chain` (*bitshares.account.Account* attribute), 98  
`chain` (*bitshares.account.AccountUpdate* attribute), 101  
`chain` (*bitshares.aio.account.Account* attribute), 20  
`chain` (*bitshares.aio.account.AccountUpdate* attribute), 23  
`chain` (*bitshares.aio.amount.Amount* attribute), 25  
`chain` (*bitshares.aio.asset.Asset* attribute), 26  
`chain` (*bitshares.aio.block.Block* attribute), 41  
`chain` (*bitshares.aio.block.BlockHeader* attribute), 43  
`chain` (*bitshares.aio.blockchain.Blockchain* attribute), 46  
`chain` (*bitshares.aio.blockchainobject.BlockchainObject* attribute), 48  
`chain` (*bitshares.aio.blockchainobject.Object* attribute), 50  
`chain` (*bitshares.aio.committee.Committee* attribute), 52  
`chain` (*bitshares.aio.dex.Dex* attribute), 54



- chain (*bitshares.aio.genesisbalance.GenesisBalance attribute*), 56
- chain (*bitshares.aio.genesisbalance.GenesisBalances attribute*), 58
- chain (*bitshares.aio.htlc.Htlc attribute*), 59
- chain (*bitshares.aio.instance.BlockchainInstance attribute*), 61
- chain (*bitshares.aio.market.Market attribute*), 63
- chain (*bitshares.aio.memo.Memo attribute*), 67
- chain (*bitshares.aio.message.Message attribute*), 69
- chain (*bitshares.aio.price.FilledOrder attribute*), 70
- chain (*bitshares.aio.price.Order attribute*), 72
- chain (*bitshares.aio.price.Price attribute*), 74
- chain (*bitshares.aio.price.PriceFeed attribute*), 76
- chain (*bitshares.aio.price.UpdateCallOrder attribute*), 77
- chain (*bitshares.aio.proposal.Proposal attribute*), 78
- chain (*bitshares.aio.proposal.Proposals attribute*), 81
- chain (*bitshares.aio.transactionbuilder.ProposalBuilder attribute*), 83
- chain (*bitshares.aio.transactionbuilder.TransactionBuilder attribute*), 84
- chain (*bitshares.aio.vesting.Vesting attribute*), 86
- chain (*bitshares.aio.wallet.Wallet attribute*), 88
- chain (*bitshares.aio.witness.Witness attribute*), 90
- chain (*bitshares.aio.witness.Witnesses attribute*), 92
- chain (*bitshares.aio.worker.Worker attribute*), 94
- chain (*bitshares.aio.worker.Workers attribute*), 96
- chain (*bitshares.amount.Amount attribute*), 103
- chain (*bitshares.asset.Asset attribute*), 105
- chain (*bitshares.block.Block attribute*), 120
- chain (*bitshares.block.BlockHeader attribute*), 122
- chain (*bitshares.blockchain.Blockchain attribute*), 125
- chain (*bitshares.blockchainobject.BlockchainObject attribute*), 127
- chain (*bitshares.blockchainobject.Object attribute*), 129
- chain (*bitshares.committee.Committee attribute*), 131
- chain (*bitshares.dex.Dex attribute*), 133
- chain (*bitshares.genesisbalance.GenesisBalance attribute*), 135
- chain (*bitshares.genesisbalance.GenesisBalances attribute*), 137
- chain (*bitshares.htlc.Htlc attribute*), 139
- chain (*bitshares.instance.BlockchainInstance attribute*), 140
- chain (*bitshares.market.Market attribute*), 143
- chain (*bitshares.memo.Memo attribute*), 147
- chain (*bitshares.message.Message attribute*), 148
- chain (*bitshares.notify.Notify attribute*), 150
- chain (*bitshares.price.FilledOrder attribute*), 151
- chain (*bitshares.price.Order attribute*), 153
- chain (*bitshares.price.Price attribute*), 155
- chain (*bitshares.price.PriceFeed attribute*), 157
- chain (*bitshares.price.UpdateCallOrder attribute*), 158
- chain (*bitshares.proposal.Proposal attribute*), 160
- chain (*bitshares.proposal.Proposals attribute*), 162
- chain (*bitshares.transactionbuilder.ProposalBuilder attribute*), 164
- chain (*bitshares.transactionbuilder.TransactionBuilder attribute*), 166
- chain (*bitshares.vesting.Vesting attribute*), 168
- chain (*bitshares.wallet.Wallet attribute*), 169
- chain (*bitshares.witness.Witness attribute*), 172
- chain (*bitshares.witness.Witnesses attribute*), 174
- chain (*bitshares.worker.Worker attribute*), 175
- chain (*bitshares.worker.Workers attribute*), 177
- chainParameters () (*bitshares.aio.blockchain.Blockchain method*), 46
- chainParameters () (*bitshares.blockchain.Blockchain method*), 125
- change\_issuer () (*bitshares.aio.asset.Asset method*), 26
- change\_issuer () (*bitshares.asset.Asset method*), 105
- changePassphrase () (*bitshares.aio.wallet.Wallet method*), 88
- changePassphrase () (*bitshares.wallet.Wallet method*), 169
- child () (*bitsharesbase.account.PrivateKey method*), 181
- child () (*bitsharesbase.account.PublicKey method*), 182
- claim () (*bitshares.aio.genesisbalance.GenesisBalance method*), 56
- claim () (*bitshares.aio.vesting.Vesting method*), 86
- claim () (*bitshares.genesisbalance.GenesisBalance method*), 135
- claim () (*bitshares.vesting.Vesting method*), 168
- claimable (*bitshares.aio.vesting.Vesting attribute*), 86
- claimable (*bitshares.vesting.Vesting attribute*), 168
- clear () (*bitshares.account.Account method*), 98
- clear () (*bitshares.account.AccountUpdate method*), 101
- clear () (*bitshares.aio.account.Account method*), 20
- clear () (*bitshares.aio.account.AccountUpdate method*), 23
- clear () (*bitshares.aio.amount.Amount method*), 25
- clear () (*bitshares.aio.asset.Asset method*), 26
- clear () (*bitshares.aio.bitshares.BitShares method*), 32
- clear () (*bitshares.aio.block.Block method*), 41
- clear () (*bitshares.aio.block.BlockHeader method*), 43
- clear () (*bitshares.aio.blockchainobject.BlockchainObject method*), 48
- clear () (*bitshares.aio.blockchainobject.Object method*), 50
- clear () (*bitshares.aio.committee.Committee method*),

52

`clear()` (*bitshares.aio.genesisbalance.GenesisBalance method*), 56

`clear()` (*bitshares.aio.genesisbalance.GenesisBalances method*), 58

`clear()` (*bitshares.aio.htlc.Htlc method*), 59

`clear()` (*bitshares.aio.market.Market method*), 63

`clear()` (*bitshares.aio.price.FilledOrder method*), 70

`clear()` (*bitshares.aio.price.Order method*), 72

`clear()` (*bitshares.aio.price.Price method*), 74

`clear()` (*bitshares.aio.price.PriceFeed method*), 76

`clear()` (*bitshares.aio.price.UpdateCallOrder method*), 77

`clear()` (*bitshares.aio.proposal.Proposal method*), 78

`clear()` (*bitshares.aio.proposal.Proposals method*), 81

`clear()` (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 84

`clear()` (*bitshares.aio.vesting.Vesting method*), 86

`clear()` (*bitshares.aio.witness.Witness method*), 90

`clear()` (*bitshares.aio.witness.Witnesses method*), 92

`clear()` (*bitshares.aio.worker.Worker method*), 94

`clear()` (*bitshares.aio.worker.Workers method*), 96

`clear()` (*bitshares.amount.Amount method*), 103

`clear()` (*bitshares.asset.Asset method*), 105

`clear()` (*bitshares.bitshares.BitShares method*), 111

`clear()` (*bitshares.block.Block method*), 120

`clear()` (*bitshares.block.BlockHeader method*), 122

`clear()` (*bitshares.blockchainobject.BlockchainObject method*), 127

`clear()` (*bitshares.blockchainobject.Object method*), 129

`clear()` (*bitshares.committee.Committee method*), 131

`clear()` (*bitshares.genesisbalance.GenesisBalance method*), 136

`clear()` (*bitshares.genesisbalance.GenesisBalances method*), 137

`clear()` (*bitshares.htlc.Htlc method*), 139

`clear()` (*bitshares.market.Market method*), 143

`clear()` (*bitshares.price.FilledOrder method*), 151

`clear()` (*bitshares.price.Order method*), 153

`clear()` (*bitshares.price.Price method*), 155

`clear()` (*bitshares.price.PriceFeed method*), 157

`clear()` (*bitshares.price.UpdateCallOrder method*), 158

`clear()` (*bitshares.proposal.Proposal method*), 160

`clear()` (*bitshares.proposal.Proposals method*), 162

`clear()` (*bitshares.transactionbuilder.TransactionBuilder method*), 166

`clear()` (*bitshares.vesting.Vesting method*), 168

`clear()` (*bitshares.witness.Witness method*), 172

`clear()` (*bitshares.witness.Witnesses method*), 174

`clear()` (*bitshares.worker.Worker method*), 175

`clear()` (*bitshares.worker.Workers method*), 177

`clear()` (*bitsharesbase.objects.AccountCreateExtensions.Buyback\_option method*), 183

`clear()` (*bitsharesbase.objects.AccountCreateExtensions.Null\_ext method*), 184

`clear()` (*bitsharesbase.objects.AccountOptions method*), 184

`clear()` (*bitsharesbase.objects.AssetOptions method*), 185

`clear()` (*bitsharesbase.objects.BitAssetOptions method*), 186

`clear()` (*bitsharesbase.objects.Memo method*), 187

`clear()` (*bitsharesbase.objects.Operation method*), 188

`clear()` (*bitsharesbase.objects.Permission method*), 189

`clear()` (*bitsharesbase.objects.Price method*), 189

`clear()` (*bitsharesbase.objects.PriceFeed method*), 190

`clear()` (*bitsharesbase.operations.Account\_create method*), 191

`clear()` (*bitsharesbase.operations.Account\_update method*), 192

`clear()` (*bitsharesbase.operations.Account\_upgrade method*), 193

`clear()` (*bitsharesbase.operations.Account\_whitelist method*), 194

`clear()` (*bitsharesbase.operations.Assert method*), 194

`clear()` (*bitsharesbase.operations.Asset\_claim\_fees method*), 195

`clear()` (*bitsharesbase.operations.Asset\_claim\_pool method*), 196

`clear()` (*bitsharesbase.operations.Asset\_create method*), 197

`clear()` (*bitsharesbase.operations.Asset\_fund\_fee\_pool method*), 197

`clear()` (*bitsharesbase.operations.Asset\_global\_settle method*), 198

`clear()` (*bitsharesbase.operations.Asset\_issue method*), 199

`clear()` (*bitsharesbase.operations.Asset\_publish\_feed method*), 199

`clear()` (*bitsharesbase.operations.Asset\_reserve method*), 200

`clear()` (*bitsharesbase.operations.Asset\_settle method*), 201

`clear()` (*bitsharesbase.operations.Asset\_update method*), 202

`clear()` (*bitsharesbase.operations.Asset\_update\_bitasset method*), 202

`clear()` (*bitsharesbase.operations.Asset\_update\_feed\_producers method*), 203

`clear()` (*bitsharesbase.operations.Asset\_update\_issuer method*), 204

`clear()` (*bitsharesbase.operations.Balance\_claim*

method), 204

clear() (bitsharesbase.operations.Bid\_collateral method), 205

clear() (bitsharesbase.operations.Call\_order\_update method), 206

clear() (bitsharesbase.operations.Committee\_member\_create method), 207

clear() (bitsharesbase.operations.Custom method), 207

clear() (bitsharesbase.operations.Htlc\_create method), 208

clear() (bitsharesbase.operations.Htlc\_extend method), 209

clear() (bitsharesbase.operations.Htlc\_redeem method), 210

clear() (bitsharesbase.operations.Limit\_order\_cancel method), 210

clear() (bitsharesbase.operations.Limit\_order\_create method), 211

clear() (bitsharesbase.operations.Liquidity\_pool\_create method), 212

clear() (bitsharesbase.operations.Liquidity\_pool\_delete method), 212

clear() (bitsharesbase.operations.Liquidity\_pool\_deposit method), 213

clear() (bitsharesbase.operations.Liquidity\_pool\_exchange method), 214

clear() (bitsharesbase.operations.Liquidity\_pool\_withdraw method), 215

clear() (bitsharesbase.operations.Op\_wrapper method), 215

clear() (bitsharesbase.operations.Override\_transfer method), 216

clear() (bitsharesbase.operations.Proposal\_create method), 217

clear() (bitsharesbase.operations.Proposal\_update method), 217

clear() (bitsharesbase.operations.Ticket\_create\_operation method), 218

clear() (bitsharesbase.operations.Ticket\_update\_operation method), 219

clear() (bitsharesbase.operations.Transfer method), 220

clear() (bitsharesbase.operations.Vesting\_balance\_withdraw method), 220

clear() (bitsharesbase.operations.Withdraw\_permission\_create method), 221

clear() (bitsharesbase.operations.Witness\_update method), 222

clear() (bitsharesbase.operations.Worker\_create method), 222

clear() (bitsharesbase.signedtransactions.Signed\_Transaction method), 224

clear\_cache() (bitshares.account.Account class method), 98

clear\_cache() (bitshares.aio.account.Account class method), 20

clear\_cache() (bitshares.aio.asset.Asset class method), 26

clear\_cache() (bitshares.aio.bitshares.BitShares class method), 32

clear\_cache() (bitshares.aio.block.Block class method), 41

clear\_cache() (bitshares.aio.block.BlockHeader class method), 43

clear\_cache() (bitshares.aio.blockchainobject.BlockchainObject class method), 48

clear\_cache() (bitshares.aio.blockchainobject.Object class method), 50

clear\_cache() (bitshares.aio.committee.Committee class method), 52

clear\_cache() (bitshares.aio.genesisbalance.GenesisBalance class method), 56

clear\_cache() (bitshares.aio.htlc.Htlc class method), 59

clear\_cache() (bitshares.aio.proposal.Proposal class method), 79

clear\_cache() (bitshares.aio.proposal.Proposals class method), 81

clear\_cache() (bitshares.aio.vesting.Vesting class method), 86

clear\_cache() (bitshares.aio.witness.Witness class method), 90

clear\_cache() (bitshares.aio.witness.Witnesses class method), 92

clear\_cache() (bitshares.aio.worker.Worker class method), 94

clear\_cache() (bitshares.aio.worker.Workers class method), 96

clear\_cache() (bitshares.asset.Asset class method), 105

clear\_cache() (bitshares.bitshares.BitShares class method), 111

clear\_cache() (bitshares.block.Block class method), 120

clear\_cache() (bitshares.block.BlockHeader class method), 122

clear\_cache() (bitshares.blockchainobject.BlockchainObject class method), 127

clear\_cache() (bitshares.blockchainobject.Object class method), 129

clear\_cache() (bitshares.committee.Committee class method), 131

clear\_cache() (bitshares.account.Account class method), 131

`shares.genesisbalance.GenesisBalance` class method), 136  
`clear_cache()` (`bitshares.htlc.Htlc` class method), 139  
`clear_cache()` (`bitshares.proposal.Proposal` class method), 160  
`clear_cache()` (`bitshares.proposal.Proposals` class method), 162  
`clear_cache()` (`bitshares.vesting.Vesting` class method), 168  
`clear_cache()` (`bitshares.witness.Witness` class method), 172  
`clear_cache()` (`bitshares.witness.Witnesses` class method), 174  
`clear_cache()` (`bitshares.worker.Worker` class method), 175  
`clear_cache()` (`bitshares.worker.Workers` class method), 177  
`close()` (`bitshares.notify.Notify` method), 150  
`close_debt_position()` (`bitshares.aio.dex.Dex` method), 54  
`close_debt_position()` (`bitshares.dex.Dex` method), 133  
`Committee` (class in `bitshares.aio.committee`), 51  
`Committee` (class in `bitshares.committee`), 130  
`Committee_member_create` (class in `bitshares-base.operations`), 207  
`compressed` (`bitsharesbase.account.PrivateKey` attribute), 181  
`compressed()` (`bitsharesbase.account.PublicKey` method), 182  
`compressed_key` (`bitsharesbase.account.PublicKey` attribute), 182  
`config` (`bitshares.aio.instance.SharedInstance` attribute), 61  
`config` (`bitshares.instance.SharedInstance` attribute), 141  
`config()` (`bitshares.aio.blockchain.Blockchain` method), 46  
`config()` (`bitshares.blockchain.Blockchain` method), 125  
`connect()` (`bitshares.aio.bitshares.BitShares` method), 32  
`connect()` (`bitshares.bitshares.BitShares` method), 111  
`constructTx()` (`bitshares.aio.transactionbuilder.TransactionBuilder` method), 84  
`constructTx()` (`bitshares.transactionbuilder.TransactionBuilder` method), 166  
`copy()` (`bitshares.account.Account` method), 98  
`copy()` (`bitshares.account.AccountUpdate` method), 101  
`copy()` (`bitshares.aio.account.Account` method), 20  
`copy()` (`bitshares.aio.account.AccountUpdate` method), 23  
`copy()` (`bitshares.aio.amount.Amount` method), 25  
`copy()` (`bitshares.aio.asset.Asset` method), 26  
`copy()` (`bitshares.aio.block.Block` method), 41  
`copy()` (`bitshares.aio.block.BlockHeader` method), 43  
`copy()` (`bitshares.aio.blockchainobject.BlockchainObject` method), 48  
`copy()` (`bitshares.aio.blockchainobject.Object` method), 50  
`copy()` (`bitshares.aio.committee.Committee` method), 52  
`copy()` (`bitshares.aio.genesisbalance.GenesisBalance` method), 56  
`copy()` (`bitshares.aio.genesisbalance.GenesisBalances` method), 58  
`copy()` (`bitshares.aio.htlc.Htlc` method), 59  
`copy()` (`bitshares.aio.market.Market` method), 64  
`copy()` (`bitshares.aio.price.FilledOrder` method), 70  
`copy()` (`bitshares.aio.price.Order` method), 72  
`copy()` (`bitshares.aio.price.Price` method), 74  
`copy()` (`bitshares.aio.price.PriceFeed` method), 76  
`copy()` (`bitshares.aio.price.UpdateCallOrder` method), 77  
`copy()` (`bitshares.aio.proposal.Proposal` method), 79  
`copy()` (`bitshares.aio.proposal.Proposals` method), 81  
`copy()` (`bitshares.aio.transactionbuilder.TransactionBuilder` method), 84  
`copy()` (`bitshares.aio.vesting.Vesting` method), 86  
`copy()` (`bitshares.aio.witness.Witness` method), 90  
`copy()` (`bitshares.aio.witness.Witnesses` method), 92  
`copy()` (`bitshares.aio.worker.Worker` method), 94  
`copy()` (`bitshares.aio.worker.Workers` method), 96  
`copy()` (`bitshares.amount.Amount` method), 103  
`copy()` (`bitshares.asset.Asset` method), 105  
`copy()` (`bitshares.block.Block` method), 120  
`copy()` (`bitshares.block.BlockHeader` method), 122  
`copy()` (`bitshares.blockchainobject.BlockchainObject` method), 127  
`copy()` (`bitshares.blockchainobject.Object` method), 129  
`copy()` (`bitshares.committee.Committee` method), 131  
`copy()` (`bitshares.genesisbalance.GenesisBalance` method), 136  
`copy()` (`bitshares.genesisbalance.GenesisBalances` method), 137  
`copy()` (`bitshares.htlc.Htlc` method), 139  
`copy()` (`bitshares.market.Market` method), 143  
`copy()` (`bitshares.price.FilledOrder` method), 151  
`copy()` (`bitshares.price.Order` method), 153  
`copy()` (`bitshares.price.Price` method), 155  
`copy()` (`bitshares.price.PriceFeed` method), 157  
`copy()` (`bitshares.price.UpdateCallOrder` method), 158  
`copy()` (`bitshares.proposal.Proposal` method), 160



- copy () (*bitshares.proposal.Proposals method*), 162
- copy () (*bitshares.transactionbuilder.TransactionBuilder method*), 166
- copy () (*bitshares.vesting.Vesting method*), 168
- copy () (*bitshares.witness.Witness method*), 172
- copy () (*bitshares.witness.Witnesses method*), 174
- copy () (*bitshares.worker.Worker method*), 175
- copy () (*bitshares.worker.Workers method*), 177
- copy () (*bitsharesbase.objects.AccountCreateExtensions.Buyback\_options method*), 183
- copy () (*bitsharesbase.objects.AccountCreateExtensions.NatContext method*), 184
- copy () (*bitsharesbase.objects.AccountOptions method*), 184
- copy () (*bitsharesbase.objects.AssetOptions method*), 185
- copy () (*bitsharesbase.objects.BitAssetOptions method*), 186
- copy () (*bitsharesbase.objects.Memo method*), 187
- copy () (*bitsharesbase.objects.Operation method*), 188
- copy () (*bitsharesbase.objects.Permission method*), 189
- copy () (*bitsharesbase.objects.Price method*), 189
- copy () (*bitsharesbase.objects.PriceFeed method*), 190
- copy () (*bitsharesbase.operations.Account\_create method*), 191
- copy () (*bitsharesbase.operations.Account\_update method*), 192
- copy () (*bitsharesbase.operations.Account\_upgrade method*), 193
- copy () (*bitsharesbase.operations.Account\_whitelist method*), 194
- copy () (*bitsharesbase.operations.Assert method*), 194
- copy () (*bitsharesbase.operations.Asset\_claim\_fees method*), 195
- copy () (*bitsharesbase.operations.Asset\_claim\_pool method*), 196
- copy () (*bitsharesbase.operations.Asset\_create method*), 197
- copy () (*bitsharesbase.operations.Asset\_fund\_fee\_pool method*), 197
- copy () (*bitsharesbase.operations.Asset\_global\_settle method*), 198
- copy () (*bitsharesbase.operations.Asset\_issue method*), 199
- copy () (*bitsharesbase.operations.Asset\_publish\_feed method*), 199
- copy () (*bitsharesbase.operations.Asset\_reserve method*), 200
- copy () (*bitsharesbase.operations.Asset\_settle method*), 201
- copy () (*bitsharesbase.operations.Asset\_update method*), 202
- copy () (*bitsharesbase.operations.Asset\_update\_bitasset method*), 202
- copy () (*bitsharesbase.operations.Asset\_update\_feed\_producers method*), 203
- copy () (*bitsharesbase.operations.Asset\_update\_issuer method*), 204
- copy () (*bitsharesbase.operations.Balance\_claim method*), 204
- copy () (*bitsharesbase.operations.Bid\_collateral method*), 205
- copy () (*bitsharesbase.operations.Call\_order\_update method*), 206
- copy () (*bitsharesbase.operations.Committee\_member\_create method*), 207
- copy () (*bitsharesbase.operations.Custom method*), 207
- copy () (*bitsharesbase.operations.Htlc\_create method*), 208
- copy () (*bitsharesbase.operations.Htlc\_extend method*), 209
- copy () (*bitsharesbase.operations.Htlc\_redeem method*), 210
- copy () (*bitsharesbase.operations.Limit\_order\_cancel method*), 210
- copy () (*bitsharesbase.operations.Limit\_order\_create method*), 211
- copy () (*bitsharesbase.operations.Liquidity\_pool\_create method*), 212
- copy () (*bitsharesbase.operations.Liquidity\_pool\_delete method*), 212
- copy () (*bitsharesbase.operations.Liquidity\_pool\_deposit method*), 213
- copy () (*bitsharesbase.operations.Liquidity\_pool\_exchange method*), 214
- copy () (*bitsharesbase.operations.Liquidity\_pool\_withdraw method*), 215
- copy () (*bitsharesbase.operations.Op\_wrapper method*), 215
- copy () (*bitsharesbase.operations.Override\_transfer method*), 216
- copy () (*bitsharesbase.operations.Proposal\_create method*), 217
- copy () (*bitsharesbase.operations.Proposal\_update method*), 217
- copy () (*bitsharesbase.operations.Ticket\_create\_operation method*), 218
- copy () (*bitsharesbase.operations.Ticket\_update\_operation method*), 219
- copy () (*bitsharesbase.operations.Transfer method*), 220
- copy () (*bitsharesbase.operations.Vesting\_balance\_withdraw method*), 220
- copy () (*bitsharesbase.operations.Withdraw\_permission\_create method*), 221
- copy () (*bitsharesbase.operations.Witness\_update method*), 222
- copy () (*bitsharesbase.operations.Worker\_create method*), 222

- method*), 222
- `copy()` (*bitsharesbase.signedtransactions.Signed\_Transaction method*), 224
- `core_base_market()` (*bitshares.aio.market.Market method*), 64
- `core_base_market()` (*bitshares.market.Market method*), 143
- `core_quote_market()` (*bitshares.aio.market.Market method*), 64
- `core_quote_market()` (*bitshares.market.Market method*), 143
- `count()` (*bitshares.aio.genesisbalance.GenesisBalances method*), 58
- `count()` (*bitshares.aio.proposal.Proposals method*), 81
- `count()` (*bitshares.aio.witness.Witnesses method*), 92
- `count()` (*bitshares.aio.worker.Workers method*), 96
- `count()` (*bitshares.genesisbalance.GenesisBalances method*), 137
- `count()` (*bitshares.proposal.Proposals method*), 162
- `count()` (*bitshares.witness.Witnesses method*), 174
- `count()` (*bitshares.worker.Workers method*), 178
- `count()` (*bitsharesbase.objects.Operation method*), 188
- `create()` (*bitshares.aio.wallet.Wallet method*), 88
- `create()` (*bitshares.wallet.Wallet method*), 170
- `create_account()` (*bitshares.aio.bitshares.BitShares method*), 32
- `create_account()` (*bitshares.bitshares.BitShares method*), 111
- `create_asset()` (*bitshares.aio.bitshares.BitShares method*), 33
- `create_asset()` (*bitshares.bitshares.BitShares method*), 112
- `create_committee_member()` (*bitshares.aio.bitshares.BitShares method*), 33
- `create_committee_member()` (*bitshares.bitshares.BitShares method*), 113
- `create_liquidity_pool()` (*bitshares.aio.bitshares.BitShares method*), 34
- `create_liquidity_pool()` (*bitshares.bitshares.BitShares method*), 113
- `create_voting_ticket()` (*bitshares.aio.bitshares.BitShares method*), 34
- `create_voting_ticket()` (*bitshares.bitshares.BitShares method*), 113
- `create_worker()` (*bitshares.aio.bitshares.BitShares method*), 34
- `create_worker()` (*bitshares.bitshares.BitShares method*), 113
- `created()` (*bitshares.aio.wallet.Wallet method*), 88
- `created()` (*bitshares.wallet.Wallet method*), 170
- `Custom` (class in *bitsharesbase.operations*), 207
- ## D
- `data` (*bitsharesbase.objects.AccountCreateExtensions.Buyback\_options attribute*), 183
- `data` (*bitsharesbase.objects.AccountCreateExtensions.Null\_ext attribute*), 184
- `data` (*bitsharesbase.objects.AccountOptions attribute*), 184
- `data` (*bitsharesbase.objects.AssetOptions attribute*), 185
- `data` (*bitsharesbase.objects.BitAssetOptions attribute*), 186
- `data` (*bitsharesbase.objects.Memo attribute*), 187
- `data` (*bitsharesbase.objects.Permission attribute*), 189
- `data` (*bitsharesbase.objects.Price attribute*), 190
- `data` (*bitsharesbase.objects.PriceFeed attribute*), 190
- `data` (*bitsharesbase.operations.Account\_create attribute*), 191
- `data` (*bitsharesbase.operations.Account\_update attribute*), 192
- `data` (*bitsharesbase.operations.Account\_upgrade attribute*), 193
- `data` (*bitsharesbase.operations.Account\_whitelist attribute*), 194
- `data` (*bitsharesbase.operations.Assert attribute*), 194
- `data` (*bitsharesbase.operations.Asset\_claim\_fees attribute*), 195
- `data` (*bitsharesbase.operations.Asset\_claim\_pool attribute*), 196
- `data` (*bitsharesbase.operations.Asset\_create attribute*), 197
- `data` (*bitsharesbase.operations.Asset\_fund\_fee\_pool attribute*), 197
- `data` (*bitsharesbase.operations.Asset\_global\_settle attribute*), 198
- `data` (*bitsharesbase.operations.Asset\_issue attribute*), 199
- `data` (*bitsharesbase.operations.Asset\_publish\_feed attribute*), 199
- `data` (*bitsharesbase.operations.Asset\_reserve attribute*), 200
- `data` (*bitsharesbase.operations.Asset\_settle attribute*), 201
- `data` (*bitsharesbase.operations.Asset\_update attribute*), 202
- `data` (*bitsharesbase.operations.Asset\_update\_bitasset attribute*), 202
- `data` (*bitsharesbase.operations.Asset\_update\_feed\_producers attribute*), 203
- `data` (*bitsharesbase.operations.Asset\_update\_issuer attribute*), 204
- `data` (*bitsharesbase.operations.Balance\_claim attribute*), 204

data (bitsharesbase.operations.Bid_collateral attribute), 205	base.signedtransactions.Signed_Transaction attribute), 224
data (bitsharesbase.operations.Call_order_update attribute), 206	define_classes() (bitshares.account.Account method), 98
data (bitsharesbase.operations.Committee_member_created attribute), 207	define_classes() (bitshares.account.AccountUpdate method), 101
data (bitsharesbase.operations.Custom attribute), 207	define_classes() (bitshares.aio.account.Account method), 20
data (bitsharesbase.operations.Htlc_create attribute), 208	define_classes() (bitshares.aio.account.AccountUpdate method), 23
data (bitsharesbase.operations.Htlc_extend attribute), 209	define_classes() (bitshares.aio.amount.Amount method), 25
data (bitsharesbase.operations.Htlc_redeem attribute), 210	define_classes() (bitshares.aio.asset.Asset method), 26
data (bitsharesbase.operations.Limit_order_cancel attribute), 210	define_classes() (bitshares.aio.bitshares.BitShares method), 34
data (bitsharesbase.operations.Limit_order_create attribute), 211	define_classes() (bitshares.aio.block.Block method), 41
data (bitsharesbase.operations.Liquidity_pool_create attribute), 212	define_classes() (bitshares.aio.block.BlockHeader method), 43
data (bitsharesbase.operations.Liquidity_pool_delete attribute), 212	define_classes() (bitshares.aio.blockchain.Blockchain method), 46
data (bitsharesbase.operations.Liquidity_pool_deposit attribute), 213	define_classes() (bitshares.aio.blockchainobject.BlockchainObject method), 48
data (bitsharesbase.operations.Liquidity_pool_exchange attribute), 214	define_classes() (bitshares.aio.blockchainobject.Object method), 50
data (bitsharesbase.operations.Liquidity_pool_withdraw attribute), 215	define_classes() (bitshares.aio.committee.Committee method), 52
data (bitsharesbase.operations.Op_wrapper attribute), 215	define_classes() (bitshares.aio.dex.Dex method), 55
data (bitsharesbase.operations.Override_transfer attribute), 216	define_classes() (bitshares.aio.genesisbalance.GenesisBalance method), 56
data (bitsharesbase.operations.Proposal_create attribute), 217	define_classes() (bitshares.aio.genesisbalance.GenesisBalances method), 58
data (bitsharesbase.operations.Proposal_update attribute), 217	define_classes() (bitshares.aio.htlc.Htlc method), 59
data (bitsharesbase.operations.Ticket_create_operation attribute), 218	define_classes() (bitshares.aio.instance.BlockchainInstance method), 61
data (bitsharesbase.operations.Ticket_update_operation attribute), 219	define_classes() (bitshares.aio.market.Market method), 64
data (bitsharesbase.operations.Transfer attribute), 220	define_classes() (bitshares.aio.memo.Memo method), 68
data (bitsharesbase.operations.Vesting_balance_withdraw attribute), 220	define_classes() (bitshares.aio.message.Message method), 69
data (bitsharesbase.operations.Withdraw_permission_create attribute), 221	
data (bitsharesbase.operations.Witness_update attribute), 222	
data (bitsharesbase.operations.Worker_create attribute), 222	
data (bitsharesbase.signedtransactions.Signed_Transaction attribute), 224	
decrypt() (bitshares.aio.memo.Memo method), 68	
decrypt() (bitshares.memo.Memo method), 147	
decrypt() (in module bitsharesbase.bip38), 182	
default_prefix (bitshares-	

<code>define_classes()</code> ( <i>bitshares.aio.price.FilledOrder method</i> ), 70	<code>define_classes()</code> ( <i>bitshares.dex.Dex method</i> ), 134
<code>define_classes()</code> ( <i>bitshares.aio.price.Order method</i> ), 72	<code>define_classes()</code> ( <i>bitshares.genesisbalance.GenesisBalance method</i> ), 136
<code>define_classes()</code> ( <i>bitshares.aio.price.Price method</i> ), 74	<code>define_classes()</code> ( <i>bitshares.genesisbalance.GenesisBalances method</i> ), 137
<code>define_classes()</code> ( <i>bitshares.aio.price.PriceFeed method</i> ), 76	<code>define_classes()</code> ( <i>bitshares.htlc.Htlc method</i> ), 139
<code>define_classes()</code> ( <i>bitshares.aio.price.UpdateCallOrder method</i> ), 77	<code>define_classes()</code> ( <i>bitshares.instance.BlockchainInstance method</i> ), 140
<code>define_classes()</code> ( <i>bitshares.aio.proposal.Proposal method</i> ), 79	<code>define_classes()</code> ( <i>bitshares.market.Market method</i> ), 143
<code>define_classes()</code> ( <i>bitshares.aio.proposal.Proposals method</i> ), 81	<code>define_classes()</code> ( <i>bitshares.memo.Memo method</i> ), 147
<code>define_classes()</code> ( <i>bitshares.aio.transactionbuilder.ProposalBuilder method</i> ), 83	<code>define_classes()</code> ( <i>bitshares.message.Message method</i> ), 148
<code>define_classes()</code> ( <i>bitshares.aio.transactionbuilder.TransactionBuilder method</i> ), 84	<code>define_classes()</code> ( <i>bitshares.notify.Notify method</i> ), 150
<code>define_classes()</code> ( <i>bitshares.aio.vesting.Vesting method</i> ), 86	<code>define_classes()</code> ( <i>bitshares.price.FilledOrder method</i> ), 151
<code>define_classes()</code> ( <i>bitshares.aio.wallet.Wallet method</i> ), 88	<code>define_classes()</code> ( <i>bitshares.price.Order method</i> ), 153
<code>define_classes()</code> ( <i>bitshares.aio.witness.Witness method</i> ), 90	<code>define_classes()</code> ( <i>bitshares.price.Price method</i> ), 156
<code>define_classes()</code> ( <i>bitshares.aio.witness.Witnesses method</i> ), 92	<code>define_classes()</code> ( <i>bitshares.price.PriceFeed method</i> ), 157
<code>define_classes()</code> ( <i>bitshares.aio.worker.Worker method</i> ), 94	<code>define_classes()</code> ( <i>bitshares.price.UpdateCallOrder method</i> ), 158
<code>define_classes()</code> ( <i>bitshares.aio.worker.Workers method</i> ), 96	<code>define_classes()</code> ( <i>bitshares.proposal.Proposal method</i> ), 160
<code>define_classes()</code> ( <i>bitshares.amount.Amount method</i> ), 103	<code>define_classes()</code> ( <i>bitshares.proposal.Proposals method</i> ), 162
<code>define_classes()</code> ( <i>bitshares.asset.Asset method</i> ), 105	<code>define_classes()</code> ( <i>bitshares.transactionbuilder.ProposalBuilder method</i> ), 164
<code>define_classes()</code> ( <i>bitshares.bitshares.BitShares method</i> ), 114	<code>define_classes()</code> ( <i>bitshares.transactionbuilder.TransactionBuilder method</i> ), 166
<code>define_classes()</code> ( <i>bitshares.block.Block method</i> ), 120	<code>define_classes()</code> ( <i>bitshares.vesting.Vesting method</i> ), 168
<code>define_classes()</code> ( <i>bitshares.block.BlockHeader method</i> ), 122	<code>define_classes()</code> ( <i>bitshares.wallet.Wallet method</i> ), 170
<code>define_classes()</code> ( <i>bitshares.blockchain.Blockchain method</i> ), 125	<code>define_classes()</code> ( <i>bitshares.witness.Witness method</i> ), 172
<code>define_classes()</code> ( <i>bitshares.blockchainobject.BlockchainObject method</i> ), 127	<code>define_classes()</code> ( <i>bitshares.witness.Witnesses method</i> ), 174
<code>define_classes()</code> ( <i>bitshares.blockchainobject.Object method</i> ), 129	<code>define_classes()</code> ( <i>bitshares.worker.Worker method</i> ), 176
<code>define_classes()</code> ( <i>bitshares.committee.Committee method</i> ), 131	<code>define_classes()</code> ( <i>bitshares.worker.Workers method</i> ), 178
	<code>delete_liquidity_pool()</code> ( <i>bitshares.aio.bitshares.BitShares method</i> ),

35  
 delete\_liquidity\_pool() (bitshares.bitshares.BitShares method), 114  
 deposit\_into\_liquidity\_pool() (bitshares.aio.bitshares.BitShares method), 35  
 deposit\_into\_liquidity\_pool() (bitshares.bitshares.BitShares method), 114  
 derive\_from\_seed() (bitsharesbase.account.PrivateKey method), 181  
 derive\_private\_key() (bitsharesbase.account.PrivateKey method), 181  
 deriveDigest() (bitsharesbase.signedtransactions.Signed\_Transaction method), 224  
 detail() (bitsharesbase.operations.Asset\_settle method), 201  
 detail() (bitsharesbase.operations.Balance\_claim method), 204  
 detail() (bitsharesbase.operations.Bid\_collateral method), 205  
 detail() (bitsharesbase.operations.Htlc\_create method), 208  
 detail() (bitsharesbase.operations.Htlc\_extend method), 209  
 detail() (bitsharesbase.operations.Htlc\_redeem method), 210  
 detail() (bitsharesbase.signedtransactions.Signed\_Transaction method), 224  
 Dex (class in bitshares.aio.dex), 53  
 Dex (class in bitshares.dex), 132  
 disableflag() (bitshares.aio.asset.Asset method), 26  
 disableflag() (bitshares.asset.Asset method), 105  
 disallow() (bitshares.aio.bitshares.BitShares method), 35  
 disallow() (bitshares.bitshares.BitShares method), 114  
 disapprovecommittee() (bitshares.aio.bitshares.BitShares method), 35  
 disapprovecommittee() (bitshares.bitshares.BitShares method), 114  
 disapproveproposal() (bitshares.aio.bitshares.BitShares method), 35  
 disapproveproposal() (bitshares.bitshares.BitShares method), 115  
 disapprovewitness() (bitshares.aio.bitshares.BitShares method), 35  
 disapprovewitness() (bitshares.bitshares.BitShares method), 115

disapproveworker() (bitshares.aio.bitshares.BitShares method), 35  
 disapproveworker() (bitshares.bitshares.BitShares method), 115

## E

elements (bitsharesbase.operations.HtlcHash attribute), 208  
 enableflag() (bitshares.aio.asset.Asset method), 27  
 enableflag() (bitshares.asset.Asset method), 105  
 encrypt() (bitshares.aio.memo.Memo method), 68  
 encrypt() (bitshares.memo.Memo method), 147  
 encrypt() (in module bitsharesbase.bip38), 182  
 ensure\_full() (bitshares.account.Account method), 98  
 ensure\_full() (bitshares.aio.account.Account method), 20  
 ensure\_full() (bitshares.aio.asset.Asset method), 27  
 ensure\_full() (bitshares.asset.Asset method), 105  
 exchange\_with\_liquidity\_pool() (bitshares.aio.bitshares.BitShares method), 36  
 exchange\_with\_liquidity\_pool() (bitshares.bitshares.BitShares method), 115  
 expiration (bitshares.aio.proposal.Proposal attribute), 79  
 expiration (bitshares.proposal.Proposal attribute), 160  
 extend() (bitshares.aio.genesisbalance.GenesisBalances method), 58  
 extend() (bitshares.aio.proposal.Proposals method), 81  
 extend() (bitshares.aio.witness.Witnesses method), 92  
 extend() (bitshares.aio.worker.Workers method), 96  
 extend() (bitshares.genesisbalance.GenesisBalances method), 138  
 extend() (bitshares.proposal.Proposals method), 162  
 extend() (bitshares.witness.Witnesses method), 174  
 extend() (bitshares.worker.Workers method), 178  
 extend() (bitsharesbase.objects.Operation method), 188  
 Extension (class in bitsharesbase.objects), 187

## F

feed (bitshares.aio.asset.Asset attribute), 27  
 feed (bitshares.asset.Asset attribute), 105  
 feeds (bitshares.aio.asset.Asset attribute), 27  
 feeds (bitshares.asset.Asset attribute), 105  
 fill\_classmaps() (in module bitsharesbase.operations), 223  
 FilledOrder (class in bitshares.aio.price), 69  
 FilledOrder (class in bitshares.price), 151



- finalizeOp() (*bitshares.aio.bitshares.BitShares method*), 36
- finalizeOp() (*bitshares.bitshares.BitShares method*), 115
- flags (*bitshares.aio.asset.Asset attribute*), 27
- flags (*bitshares.asset.Asset attribute*), 105
- for\_sale (*bitshares.price.Order attribute*), 153
- force\_flag() (in module *bitshares-base.asset\_permissions*), 182
- from\_privkey() (*bitsharesbase.account.PublicKey class method*), 182
- from\_pubkey() (*bitsharesbase.account.Address class method*), 179
- fromkeys() (*bitshares.account.Account method*), 98
- fromkeys() (*bitshares.account.AccountUpdate method*), 101
- fromkeys() (*bitshares.aio.account.Account method*), 20
- fromkeys() (*bitshares.aio.account.AccountUpdate method*), 23
- fromkeys() (*bitshares.aio.amount.Amount method*), 25
- fromkeys() (*bitshares.aio.asset.Asset method*), 27
- fromkeys() (*bitshares.aio.block.Block method*), 41
- fromkeys() (*bitshares.aio.block.BlockHeader method*), 43
- fromkeys() (*bitshares.aio.blockchainobject.BlockchainObject method*), 48
- fromkeys() (*bitshares.aio.blockchainobject.Object method*), 50
- fromkeys() (*bitshares.aio.committee.Committee method*), 52
- fromkeys() (*bitshares.aio.genesisbalance.GenesisBalance method*), 56
- fromkeys() (*bitshares.aio.htlc.Htlc method*), 59
- fromkeys() (*bitshares.aio.market.Market method*), 64
- fromkeys() (*bitshares.aio.price.FilledOrder method*), 70
- fromkeys() (*bitshares.aio.price.Order method*), 72
- fromkeys() (*bitshares.aio.price.Price method*), 74
- fromkeys() (*bitshares.aio.price.PriceFeed method*), 76
- fromkeys() (*bitshares.aio.price.UpdateCallOrder method*), 77
- fromkeys() (*bitshares.aio.proposal.Proposal method*), 79
- fromkeys() (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 84
- fromkeys() (*bitshares.aio.vesting.Vesting method*), 86
- fromkeys() (*bitshares.aio.witness.Witness method*), 90
- fromkeys() (*bitshares.aio.worker.Worker method*), 94
- fromkeys() (*bitshares.amount.Amount method*), 103
- fromkeys() (*bitshares.asset.Asset method*), 105
- fromkeys() (*bitshares.block.Block method*), 120
- fromkeys() (*bitshares.block.BlockHeader method*), 122
- fromkeys() (*bitshares.blockchainobject.BlockchainObject method*), 127
- fromkeys() (*bitshares.blockchainobject.Object method*), 129
- fromkeys() (*bitshares.committee.Committee method*), 131
- fromkeys() (*bitshares.genesisbalance.GenesisBalance method*), 136
- fromkeys() (*bitshares.htlc.Htlc method*), 139
- fromkeys() (*bitshares.market.Market method*), 143
- fromkeys() (*bitshares.price.FilledOrder method*), 151
- fromkeys() (*bitshares.price.Order method*), 153
- fromkeys() (*bitshares.price.Price method*), 156
- fromkeys() (*bitshares.price.PriceFeed method*), 157
- fromkeys() (*bitshares.price.UpdateCallOrder method*), 158
- fromkeys() (*bitshares.proposal.Proposal method*), 160
- fromkeys() (*bitshares.transactionbuilder.TransactionBuilder method*), 166
- fromkeys() (*bitshares.vesting.Vesting method*), 168
- fromkeys() (*bitshares.witness.Witness method*), 172
- fromkeys() (*bitshares.worker.Worker method*), 176
- fromkeys() (*bitshares-base.objects.AccountCreateExtensions.Buyback\_options method*), 183
- fromkeys() (*bitshares-base.objects.AccountCreateExtensions.Null\_ext method*), 184
- fromkeys() (*bitsharesbase.objects.AccountOptions method*), 185
- fromkeys() (*bitsharesbase.objects.AssetOptions method*), 185
- fromkeys() (*bitsharesbase.objects.BitAssetOptions method*), 186
- fromkeys() (*bitsharesbase.objects.Memo method*), 187
- fromkeys() (*bitsharesbase.objects.Permission method*), 189
- fromkeys() (*bitsharesbase.objects.Price method*), 190
- fromkeys() (*bitsharesbase.objects.PriceFeed method*), 190
- fromkeys() (*bitsharesbase.operations.Account\_create method*), 191
- fromkeys() (*bitshares-base.operations.Account\_update method*), 192
- fromkeys() (*bitshares-base.operations.Account\_upgrade method*), 193
- fromkeys() (*bitshares-*

[base.operations.Account\\_whitelist method](#)), 194  
[fromkeys\(\)](#) ([bitsharesbase.operations.Assert method](#)), 194  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_claim\\_fees method](#)), 195  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_claim\\_pool method](#)), 196  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_create method](#)), 197  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_fund\\_fee\\_pool method](#)), 197  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_global\\_settle method](#)), 198  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_issue method](#)), 199  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_publish\\_feed method](#)), 199  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_reserve method](#)), 200  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_settle method](#)), 201  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_update method](#)), 202  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_update\\_bitasset method](#)), 202  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_update\\_feed\\_producers method](#)), 203  
[fromkeys\(\)](#) ([bitsharesbase.operations.Asset\\_update\\_issuer method](#)), 204  
[fromkeys\(\)](#) ([bitsharesbase.operations.Balance\\_claim method](#)), 204  
[fromkeys\(\)](#) ([bitsharesbase.operations.Bid\\_collateral method](#)), 205  
[fromkeys\(\)](#) ([bitsharesbase.operations.Call\\_order\\_update method](#)), 206  
[fromkeys\(\)](#) ([bitsharesbase.operations.Committee\\_member\\_create method](#)), 207  
[fromkeys\(\)](#) ([bitsharesbase.operations.Custom method](#)), 207  
[fromkeys\(\)](#) ([bitsharesbase.operations.Htlc\\_create method](#)), 208  
[fromkeys\(\)](#) ([bitsharesbase.operations.Htlc\\_extend method](#)), 209  
[fromkeys\(\)](#) ([bitsharesbase.operations.Htlc\\_redeem method](#)), 210  
[fromkeys\(\)](#) ([bitsharesbase.operations.Limit\\_order\\_cancel method](#)), 210  
[fromkeys\(\)](#) ([bitsharesbase.operations.Limit\\_order\\_create method](#)), 211  
[fromkeys\(\)](#) ([bitsharesbase.operations.Liquidity\\_pool\\_create method](#)), 212  
[fromkeys\(\)](#) ([bitsharesbase.operations.Liquidity\\_pool\\_delete method](#)), 213  
[fromkeys\(\)](#) ([bitsharesbase.operations.Liquidity\\_pool\\_deposit method](#)), 213  
[fromkeys\(\)](#) ([bitsharesbase.operations.Liquidity\\_pool\\_exchange method](#)), 214  
[fromkeys\(\)](#) ([bitsharesbase.operations.Liquidity\\_pool\\_withdraw method](#)), 215  
[fromkeys\(\)](#) ([bitsharesbase.operations.Op\\_wrapper method](#)), 215  
[fromkeys\(\)](#) ([bitsharesbase.operations.Override\\_transfer method](#)), 216  
[fromkeys\(\)](#) ([bitsharesbase.operations.Proposal\\_create method](#)), 217  
[fromkeys\(\)](#) ([bitsharesbase.operations.Proposal\\_update method](#)), 218  
[fromkeys\(\)](#) ([bitsharesbase.operations.Ticket\\_create\\_operation method](#)), 218  
[fromkeys\(\)](#) ([bitsharesbase.operations.Ticket\\_update\\_operation method](#)), 219  
[fromkeys\(\)](#) ([bitsharesbase.operations.Transfer method](#)), 220  
[fromkeys\(\)](#) ([bitsharesbase.operations.Vesting\\_balance\\_withdraw method](#)), 220  
[fromkeys\(\)](#) ([bitsharesbase.operations.Withdraw\\_permission\\_create method](#)), 221  
[fromkeys\(\)](#) ([bitsharesbase.operations.Witness\\_update method](#)), 222  
[fromkeys\(\)](#) ([bitsharesbase.operations.Worker\\_create method](#)), 223  
[fromkeys\(\)](#) ([bitsharesbase.operations.Xrp\\_claim method](#)), 223

*base.signedtransactions.Signed\_Transaction method*), 224  
fromlist (*bitsharesbase.objects.Operation attribute*), 188  
fund\_fee\_pool() (*bitshares.aio.bitshares.BitShares method*), 36  
fund\_fee\_pool() (*bitshares.bitshares.BitShares method*), 116

## G

GenesisBalance (class in *bitshares.aio.genesisbalance*), 55  
GenesisBalance (class in *bitshares.genesisbalance*), 135  
GenesisBalances (class in *bitshares.aio.genesisbalance*), 57  
GenesisBalances (class in *bitshares.genesisbalance*), 137  
get() (*bitshares.account.Account method*), 98  
get() (*bitshares.account.AccountUpdate method*), 101  
get() (*bitshares.aio.account.Account method*), 20  
get() (*bitshares.aio.account.AccountUpdate method*), 23  
get() (*bitshares.aio.amount.Amount method*), 25  
get() (*bitshares.aio.asset.Asset method*), 27  
get() (*bitshares.aio.block.Block method*), 41  
get() (*bitshares.aio.block.BlockHeader method*), 43  
get() (*bitshares.aio.blockchainobject.BlockchainObject method*), 48  
get() (*bitshares.aio.blockchainobject.Object method*), 50  
get() (*bitshares.aio.committee.Committee method*), 52  
get() (*bitshares.aio.genesisbalance.GenesisBalance method*), 56  
get() (*bitshares.aio.htlc.Htlc method*), 59  
get() (*bitshares.aio.market.Market method*), 64  
get() (*bitshares.aio.price.FilledOrder method*), 70  
get() (*bitshares.aio.price.Order method*), 72  
get() (*bitshares.aio.price.Price method*), 74  
get() (*bitshares.aio.price.PriceFeed method*), 76  
get() (*bitshares.aio.price.UpdateCallOrder method*), 77  
get() (*bitshares.aio.proposal.Proposal method*), 79  
get() (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 84  
get() (*bitshares.aio.vesting.Vesting method*), 86  
get() (*bitshares.aio.witness.Witness method*), 90  
get() (*bitshares.aio.worker.Worker method*), 94  
get() (*bitshares.amount.Amount method*), 103  
get() (*bitshares.asset.Asset method*), 105  
get() (*bitshares.block.Block method*), 120  
get() (*bitshares.block.BlockHeader method*), 122  
get() (*bitshares.blockchainobject.BlockchainObject method*), 127

get() (*bitshares.blockchainobject.Object method*), 129  
get() (*bitshares.committee.Committee method*), 131  
get() (*bitshares.genesisbalance.GenesisBalance method*), 136  
get() (*bitshares.htlc.Htlc method*), 139  
get() (*bitshares.market.Market method*), 143  
get() (*bitshares.price.FilledOrder method*), 151  
get() (*bitshares.price.Order method*), 153  
get() (*bitshares.price.Price method*), 156  
get() (*bitshares.price.PriceFeed method*), 157  
get() (*bitshares.price.UpdateCallOrder method*), 158  
get() (*bitshares.proposal.Proposal method*), 160  
get() (*bitshares.transactionbuilder.TransactionBuilder method*), 166  
get() (*bitshares.vesting.Vesting method*), 168  
get() (*bitshares.witness.Witness method*), 172  
get() (*bitshares.worker.Worker method*), 176  
get() (*bitsharesbase.objects.AccountCreateExtensions.Buyback\_options method*), 183  
get() (*bitsharesbase.objects.AccountCreateExtensions.Null\_ext method*), 184  
get() (*bitsharesbase.objects.AccountOptions method*), 185  
get() (*bitsharesbase.objects.AssetOptions method*), 185  
get() (*bitsharesbase.objects.BitAssetOptions method*), 186  
get() (*bitsharesbase.objects.Memo method*), 187  
get() (*bitsharesbase.objects.Permission method*), 189  
get() (*bitsharesbase.objects.Price method*), 190  
get() (*bitsharesbase.objects.PriceFeed method*), 190  
get() (*bitsharesbase.operations.Account\_create method*), 192  
get() (*bitsharesbase.operations.Account\_update method*), 192  
get() (*bitsharesbase.operations.Account\_upgrade method*), 193  
get() (*bitsharesbase.operations.Account\_whitelist method*), 194  
get() (*bitsharesbase.operations.Assert method*), 195  
get() (*bitsharesbase.operations.Asset\_claim\_fees method*), 195  
get() (*bitsharesbase.operations.Asset\_claim\_pool method*), 196  
get() (*bitsharesbase.operations.Asset\_create method*), 197  
get() (*bitsharesbase.operations.Asset\_fund\_fee\_pool method*), 197  
get() (*bitsharesbase.operations.Asset\_global\_settle method*), 198  
get() (*bitsharesbase.operations.Asset\_issue method*), 199  
get() (*bitsharesbase.operations.Asset\_publish\_feed method*), 200



get () (bitsharesbase.operations.Asset_reserve method), 200	get () (bitsharesbase.operations.Vesting_balance_withdraw method), 220
get () (bitsharesbase.operations.Asset_settle method), 201	get () (bitsharesbase.operations.Withdraw_permission_create method), 221
get () (bitsharesbase.operations.Asset_update method), 202	get () (bitsharesbase.operations.Witness_update method), 222
get () (bitsharesbase.operations.Asset_update_bitasset method), 202	get () (bitsharesbase.operations.Worker_create method), 223
get () (bitsharesbase.operations.Asset_update_feed_producers method), 203	get () (bitsharesbase.signedtransactions.Signed_Transaction method), 224
get () (bitsharesbase.operations.Asset_update_issuer method), 204	get_all_accounts () (bitshares.aio.blockchain.Blockchain method), 46
get () (bitsharesbase.operations.Balance_claim method), 205	get_all_accounts () (bitshares.blockchain.Blockchain method), 125
get () (bitsharesbase.operations.Bid_collateral method), 205	get_blind_private () (bitsharesbase.account.BrainKey method), 180
get () (bitsharesbase.operations.Call_order_update method), 206	get_block_interval () (bitshares.aio.blockchain.Blockchain method), 46
get () (bitsharesbase.operations.Committee_member_create method), 207	get_block_interval () (bitshares.blockchain.Blockchain method), 125
get () (bitsharesbase.operations.Custom method), 207	get_block_params () (bitshares.aio.transactionbuilder.TransactionBuilder method), 84
get () (bitsharesbase.operations.Htlc_create method), 208	get_block_params () (bitshares.transactionbuilder.TransactionBuilder method), 166
get () (bitsharesbase.operations.Htlc_extend method), 209	get_brainkey () (bitsharesbase.account.BrainKey method), 180
get () (bitsharesbase.operations.Htlc_redeem method), 210	get_call_orders () (bitshares.aio.asset.Asset method), 27
get () (bitsharesbase.operations.Limit_order_cancel method), 210	get_call_orders () (bitshares.asset.Asset method), 105
get () (bitsharesbase.operations.Limit_order_create method), 211	get_chain_properties () (bitshares.aio.blockchain.Blockchain method), 46
get () (bitsharesbase.operations.Liquidity_pool_create method), 212	get_chain_properties () (bitshares.blockchain.Blockchain method), 125
get () (bitsharesbase.operations.Liquidity_pool_delete method), 213	get_current_block () (bitshares.aio.blockchain.Blockchain method), 46
get () (bitsharesbase.operations.Liquidity_pool_deposit method), 213	get_current_block () (bitshares.blockchain.Blockchain method), 125
get () (bitsharesbase.operations.Liquidity_pool_exchange method), 214	get_current_block_num () (bitshares.aio.blockchain.Blockchain method), 46
get () (bitsharesbase.operations.Liquidity_pool_withdraw method), 215	get_current_block_num () (bitshares.blockchain.Blockchain method), 125
get () (bitsharesbase.operations.Op_wrapper method), 215	get_default_config_store () (in module bitshares.storage), 163
get () (bitsharesbase.operations.Override_transfer method), 216	get_default_key_store () (in module bitshares.storage), 163
get () (bitsharesbase.operations.Proposal_create method), 217	get_default_prefix () (bitshares-
get () (bitsharesbase.operations.Proposal_update method), 218	
get () (bitsharesbase.operations.Ticket_create_operation method), 218	
get () (bitsharesbase.operations.Ticket_update_operation method), 219	
get () (bitsharesbase.operations.Transfer method), 220	

*base.signedtransactions.Signed\_Transaction method), 224*

*get\_instance\_class()* (*bitshares.account.Account method), 99*

*get\_instance\_class()* (*bitshares.account.AccountUpdate method), 101*

*get\_instance\_class()* (*bitshares.aio.account.Account method), 20*

*get\_instance\_class()* (*bitshares.aio.account.AccountUpdate method), 23*

*get\_instance\_class()* (*bitshares.aio.amount.Amount method), 25*

*get\_instance\_class()* (*bitshares.aio.asset.Asset method), 27*

*get\_instance\_class()* (*bitshares.aio.block.Block method), 41*

*get\_instance\_class()* (*bitshares.aio.block.BlockHeader method), 43*

*get\_instance\_class()* (*bitshares.aio.blockchain.Blockchain method), 46*

*get\_instance\_class()* (*bitshares.aio.blockchainobject.BlockchainObject method), 48*

*get\_instance\_class()* (*bitshares.aio.blockchainobject.Object method), 50*

*get\_instance\_class()* (*bitshares.aio.committee.Committee method), 52*

*get\_instance\_class()* (*bitshares.aio.dex.Dex method), 55*

*get\_instance\_class()* (*bitshares.aio.genesisbalance.GenesisBalance method), 56*

*get\_instance\_class()* (*bitshares.aio.genesisbalance.GenesisBalances method), 58*

*get\_instance\_class()* (*bitshares.aio.htlc.Htlc method), 59*

*get\_instance\_class()* (*bitshares.aio.instance.BlockchainInstance method), 61*

*get\_instance\_class()* (*bitshares.aio.market.Market method), 64*

*get\_instance\_class()* (*bitshares.aio.memo.Memo method), 68*

*get\_instance\_class()* (*bitshares.aio.message.Message method), 69*

*get\_instance\_class()* (*bitshares.aio.price.FilledOrder method), 70*

*get\_instance\_class()* (*bitshares.aio.price.Order method), 72*

*get\_instance\_class()* (*bitshares.aio.price.Price method), 74*

*get\_instance\_class()* (*bitshares.aio.price.PriceFeed method), 76*

*get\_instance\_class()* (*bitshares.aio.price.UpdateCallOrder method), 77*

*get\_instance\_class()* (*bitshares.aio.proposal.Proposal method), 79*

*get\_instance\_class()* (*bitshares.aio.proposal.Proposals method), 81*

*get\_instance\_class()* (*bitshares.aio.transactionbuilder.ProposalBuilder method), 83*

*get\_instance\_class()* (*bitshares.aio.transactionbuilder.TransactionBuilder method), 84*

*get\_instance\_class()* (*bitshares.aio.vesting.Vesting method), 86*

*get\_instance\_class()* (*bitshares.aio.wallet.Wallet method), 89*

*get\_instance\_class()* (*bitshares.aio.witness.Witness method), 90*

*get\_instance\_class()* (*bitshares.aio.witness.Witnesses method), 92*

*get\_instance\_class()* (*bitshares.aio.worker.Worker method), 94*

*get\_instance\_class()* (*bitshares.aio.worker.Workers method), 96*

*get\_instance\_class()* (*bitshares.amount.Amount method), 103*

*get\_instance\_class()* (*bitshares.asset.Asset method), 105*

*get\_instance\_class()* (*bitshares.block.Block method), 120*

*get\_instance\_class()* (*bitshares.block.BlockHeader method), 122*

*get\_instance\_class()* (*bitshares.blockchain.Blockchain method), 125*

*get\_instance\_class()* (*bitshares.blockchainobject.BlockchainObject method), 127*

*get\_instance\_class()* (*bitshares.blockchainobject.Object method), 129*

*get\_instance\_class()* (*bitshares.committee.Committee method), 131*

*get\_instance\_class()* (*bitshares.dex.Dex method), 134*

*get\_instance\_class()* (*bitshares.genesisbalance.GenesisBalance method), 136*

`get_instance_class()` (*bitshares.genesisbalance.GenesisBalances method*), 138  
`get_instance_class()` (*bitshares.htlc.Htlc method*), 139  
`get_instance_class()` (*bitshares.instance.BlockchainInstance method*), 141  
`get_instance_class()` (*bitshares.market.Market method*), 143  
`get_instance_class()` (*bitshares.memo.Memo method*), 148  
`get_instance_class()` (*bitshares.message.Message method*), 148  
`get_instance_class()` (*bitshares.notify.Notify method*), 150  
`get_instance_class()` (*bitshares.price.FilledOrder method*), 151  
`get_instance_class()` (*bitshares.price.Order method*), 153  
`get_instance_class()` (*bitshares.price.Price method*), 156  
`get_instance_class()` (*bitshares.price.PriceFeed method*), 157  
`get_instance_class()` (*bitshares.price.UpdateCallOrder method*), 159  
`get_instance_class()` (*bitshares.proposal.Proposal method*), 160  
`get_instance_class()` (*bitshares.proposal.Proposals method*), 162  
`get_instance_class()` (*bitshares.transactionbuilder.ProposalBuilder method*), 164  
`get_instance_class()` (*bitshares.transactionbuilder.TransactionBuilder method*), 166  
`get_instance_class()` (*bitshares.vesting.Vesting method*), 168  
`get_instance_class()` (*bitshares.wallet.Wallet method*), 170  
`get_instance_class()` (*bitshares.witness.Witness method*), 172  
`get_instance_class()` (*bitshares.witness.Witnesses method*), 174  
`get_instance_class()` (*bitshares.worker.Worker method*), 176  
`get_instance_class()` (*bitshares.worker.Workers method*), 178  
`get_limit_orders()` (*bitshares.aio.market.Market method*), 64  
`get_limit_orders()` (*bitshares.market.Market method*), 144  
`get_market_ids()` (*bitshares.notify.Notify method*), 150  
`get_network()` (*bitshares.aio.blockchain.Blockchain method*), 46  
`get_network()` (*bitshares.blockchain.Blockchain method*), 125  
`get_parent()` (*bitshares.aio.transactionbuilder.ProposalBuilder method*), 83  
`get_parent()` (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 84  
`get_parent()` (*bitshares.transactionbuilder.ProposalBuilder method*), 164  
`get_parent()` (*bitshares.transactionbuilder.TransactionBuilder method*), 166  
`get_private()` (*bitsharesbase.account.BrainKey method*), 180  
`get_private()` (*bitsharesbase.account.PasswordKey method*), 180  
`get_private_key()` (*bitsharesbase.account.BrainKey method*), 180  
`get_private_key()` (*bitsharesbase.account.PasswordKey method*), 180  
`get_public()` (*bitsharesbase.account.BrainKey method*), 180  
`get_public()` (*bitsharesbase.account.PasswordKey method*), 180  
`get_public_key()` (*bitsharesbase.account.BrainKey method*), 180  
`get_public_key()` (*bitsharesbase.account.PasswordKey method*), 180  
`get_raw()` (*bitshares.aio.transactionbuilder.ProposalBuilder method*), 83  
`get_raw()` (*bitshares.transactionbuilder.ProposalBuilder method*), 164  
`get_secret()` (*bitsharesbase.account.PrivateKey method*), 181  
`get_settle_orders()` (*bitshares.aio.asset.Asset method*), 27  
`get_settle_orders()` (*bitshares.asset.Asset method*), 105  
`get_string()` (*bitshares.aio.market.Market method*), 64  
`get_string()` (*bitshares.market.Market method*), 144  
`getAccountFromPrivateKey()` (*bitshares.aio.wallet.Wallet method*), 88  
`getAccountFromPrivateKey()` (*bitshares.wallet.Wallet method*), 170  
`getAccountFromPublicKey()` (*bitshares.aio.wallet.Wallet method*), 88  
`getAccountFromPublicKey()` (*bitshares.wallet.Wallet method*), 170  
`getAccounts()` (*bitshares.aio.wallet.Wallet method*), 88  
`getAccounts()` (*bitshares.wallet.Wallet method*), 170  
`getAccountsFromPublicKey()` (*bit-*

*shares.aio.wallet.Wallet method*), 88  
 getAccountsFromPublicKey() (*bitshares.aio.wallet.Wallet method*), 170  
 getActiveKeyForAccount() (*bitshares.aio.wallet.Wallet method*), 88  
 getActiveKeyForAccount() (*bitshares.wallet.Wallet method*), 170  
 getAllAccounts() (*bitshares.aio.wallet.Wallet method*), 88  
 getAllAccounts() (*bitshares.wallet.Wallet method*), 170  
 getChainParams() (*bitshares-base.signedtransactions.Signed\_Transaction method*), 224  
 getfromcache() (*bitshares.account.Account method*), 99  
 getfromcache() (*bitshares.aio.account.Account method*), 20  
 getfromcache() (*bitshares.aio.asset.Asset method*), 27  
 getfromcache() (*bitshares.aio.block.Block method*), 41  
 getfromcache() (*bitshares.aio.block.BlockHeader method*), 43  
 getfromcache() (*bitshares.aio.blockchainobject.BlockchainObject method*), 48  
 getfromcache() (*bitshares.aio.blockchainobject.Object method*), 50  
 getfromcache() (*bitshares.aio.committee.Committee method*), 52  
 getfromcache() (*bitshares.aio.genesisbalance.GenesisBalance method*), 56  
 getfromcache() (*bitshares.aio.htlc.Htlc method*), 59  
 getfromcache() (*bitshares.aio.proposal.Proposal method*), 79  
 getfromcache() (*bitshares.aio.proposal.Proposals method*), 81  
 getfromcache() (*bitshares.aio.vesting.Vesting method*), 86  
 getfromcache() (*bitshares.aio.witness.Witness method*), 90  
 getfromcache() (*bitshares.aio.witness.Witnesses method*), 92  
 getfromcache() (*bitshares.aio.worker.Worker method*), 94  
 getfromcache() (*bitshares.aio.worker.Workers method*), 96  
 getfromcache() (*bitshares.asset.Asset method*), 105  
 getfromcache() (*bitshares.block.Block method*), 120  
 getfromcache() (*bitshares.block.BlockHeader method*), 122  
 getfromcache() (*bitshares.blockchainobject.BlockchainObject method*), 127  
 getfromcache() (*bitshares.blockchainobject.Object method*), 129  
 getfromcache() (*bitshares.committee.Committee method*), 131  
 getfromcache() (*bitshares.genesisbalance.GenesisBalance method*), 136  
 getfromcache() (*bitshares.htlc.Htlc method*), 139  
 getfromcache() (*bitshares.proposal.Proposal method*), 160  
 getfromcache() (*bitshares.proposal.Proposals method*), 162  
 getfromcache() (*bitshares.vesting.Vesting method*), 168  
 getfromcache() (*bitshares.witness.Witness method*), 172  
 getfromcache() (*bitshares.witness.Witnesses method*), 174  
 getfromcache() (*bitshares.worker.Worker method*), 176  
 getfromcache() (*bitshares.worker.Workers method*), 178  
 getKeyType() (*bitshares.aio.wallet.Wallet method*), 88  
 getKeyType() (*bitshares.wallet.Wallet method*), 170  
 getKnownChains() (*bitshares-base.signedtransactions.Signed\_Transaction method*), 224  
 getMemoKeyForAccount() (*bitshares.aio.wallet.Wallet method*), 88  
 getMemoKeyForAccount() (*bitshares.wallet.Wallet method*), 170  
 getOperationClassForId() (in module *bitsharesbase.operations*), 223  
 getOperationIdForClass() (in module *bitsharesbase.operations*), 223  
 getOperationIdForName() (*bitshares-base.objects.Operation method*), 188  
 getOperationKlass() (*bitshares-base.signedtransactions.Signed\_Transaction method*), 224  
 getOperationName() (in module *bitshares-base.operationids*), 191  
 getOperationNameForId() (*bitshares-base.objects.Operation method*), 188  
 getOperationNameForId() (in module *bitshares-base.operationids*), 191  
 getOperationNameForId() (in module *bitshares-base.operations*), 223  
 getOwnerKeyForAccount() (*bit-*

*shares.aio.wallet.Wallet method*), 88  
 getOwnerKeyForAccount() (*bitshares.wallet.Wallet method*), 170  
 getPrivateKeyForPublicKey() (*bitshares.aio.wallet.Wallet method*), 88  
 getPrivateKeyForPublicKey() (*bitshares.wallet.Wallet method*), 170  
 getPublicKeys() (*bitshares.aio.wallet.Wallet method*), 88  
 getPublicKeys() (*bitshares.wallet.Wallet method*), 170

## H

halt() (*bitshares.aio.asset.Asset method*), 27  
 halt() (*bitshares.asset.Asset method*), 105  
 history() (*bitshares.account.Account method*), 99  
 history() (*bitshares.aio.account.Account method*), 21  
 Htlc (*class in bitshares.aio.htlc*), 59  
 Htlc (*class in bitshares.htlc*), 138  
 Htlc\_create (*class in bitsharesbase.operations*), 208  
 htlc\_create() (*bitshares.aio.bitshares.BitShares method*), 36  
 htlc\_create() (*bitshares.bitshares.BitShares method*), 116  
 Htlc\_extend (*class in bitsharesbase.operations*), 209  
 Htlc\_redeem (*class in bitsharesbase.operations*), 209  
 htlc\_redeem() (*bitshares.aio.bitshares.BitShares method*), 37  
 htlc\_redeem() (*bitshares.bitshares.BitShares method*), 116  
 HtlcDoesNotExistException, 134  
 HtlcHash (*class in bitsharesbase.operations*), 208

## I

id (*bitsharesbase.objects.Operation attribute*), 188  
 id (*bitsharesbase.signedtransactions.Signed\_Transaction attribute*), 224  
 identifier (*bitshares.account.Account attribute*), 99  
 identifier (*bitshares.aio.account.Account attribute*), 21  
 identifier (*bitshares.aio.asset.Asset attribute*), 27  
 identifier (*bitshares.aio.block.Block attribute*), 42  
 identifier (*bitshares.aio.block.BlockHeader attribute*), 43  
 identifier (*bitshares.aio.blockchainobject.BlockchainObject attribute*), 48  
 identifier (*bitshares.aio.blockchainobject.Object attribute*), 50  
 identifier (*bitshares.aio.committee.Committee attribute*), 52  
 identifier (*bitshares.aio.genesisbalance.GenesisBalance attribute*), 56  
 identifier (*bitshares.aio.htlc.Htlc attribute*), 59

identifier (*bitshares.aio.proposal.Proposal attribute*), 79  
 identifier (*bitshares.aio.proposal.Proposals attribute*), 81  
 identifier (*bitshares.aio.vesting.Vesting attribute*), 86  
 identifier (*bitshares.aio.witness.Witness attribute*), 90  
 identifier (*bitshares.aio.witness.Witnesses attribute*), 92  
 identifier (*bitshares.aio.worker.Worker attribute*), 94  
 identifier (*bitshares.aio.worker.Workers attribute*), 96  
 identifier (*bitshares.asset.Asset attribute*), 106  
 identifier (*bitshares.block.Block attribute*), 120  
 identifier (*bitshares.block.BlockHeader attribute*), 122  
 identifier (*bitshares.blockchainobject.BlockchainObject attribute*), 127  
 identifier (*bitshares.blockchainobject.Object attribute*), 129  
 identifier (*bitshares.committee.Committee attribute*), 131  
 identifier (*bitshares.genesisbalance.GenesisBalance attribute*), 136  
 identifier (*bitshares.htlc.Htlc attribute*), 139  
 identifier (*bitshares.proposal.Proposal attribute*), 160  
 identifier (*bitshares.proposal.Proposals attribute*), 162  
 identifier (*bitshares.vesting.Vesting attribute*), 168  
 identifier (*bitshares.witness.Witness attribute*), 172  
 identifier (*bitshares.witness.Witnesses attribute*), 174  
 identifier (*bitshares.worker.Worker attribute*), 176  
 identifier (*bitshares.worker.Workers attribute*), 178  
 incached() (*bitshares.account.Account method*), 99  
 incached() (*bitshares.aio.account.Account method*), 21  
 incached() (*bitshares.aio.asset.Asset method*), 27  
 incached() (*bitshares.aio.block.Block method*), 42  
 incached() (*bitshares.aio.block.BlockHeader method*), 43  
 incached() (*bitshares.aio.blockchainobject.BlockchainObject method*), 48  
 incached() (*bitshares.aio.blockchainobject.Object method*), 50  
 incached() (*bitshares.aio.committee.Committee method*), 52  
 incached() (*bitshares.aio.genesisbalance.GenesisBalance method*), 56  
 incached() (*bitshares.aio.htlc.Htlc method*), 59  
 incached() (*bitshares.aio.proposal.Proposal*



[method](#)), 79

[incached\(\)](#) ([bitshares.aio.proposal.Proposals method](#)), 81

[incached\(\)](#) ([bitshares.aio.vesting.Vesting method](#)), 86

[incached\(\)](#) ([bitshares.aio.witness.Witness method](#)), 90

[incached\(\)](#) ([bitshares.aio.witness.Witnesses method](#)), 92

[incached\(\)](#) ([bitshares.aio.worker.Worker method](#)), 94

[incached\(\)](#) ([bitshares.aio.worker.Workers method](#)), 96

[incached\(\)](#) ([bitshares.asset.Asset method](#)), 106

[incached\(\)](#) ([bitshares.block.Block method](#)), 121

[incached\(\)](#) ([bitshares.block.BlockHeader method](#)), 122

[incached\(\)](#) ([bitshares.blockchainobject.BlockchainObject method](#)), 127

[incached\(\)](#) ([bitshares.blockchainobject.Object method](#)), 129

[incached\(\)](#) ([bitshares.committee.Committee method](#)), 131

[incached\(\)](#) ([bitshares.genesisbalance.GenesisBalance method](#)), 136

[incached\(\)](#) ([bitshares.htlc.Htlc method](#)), 139

[incached\(\)](#) ([bitshares.proposal.Proposal method](#)), 160

[incached\(\)](#) ([bitshares.proposal.Proposals method](#)), 162

[incached\(\)](#) ([bitshares.vesting.Vesting method](#)), 168

[incached\(\)](#) ([bitshares.witness.Witness method](#)), 172

[incached\(\)](#) ([bitshares.witness.Witnesses method](#)), 174

[incached\(\)](#) ([bitshares.worker.Worker method](#)), 176

[incached\(\)](#) ([bitshares.worker.Workers method](#)), 178

[index\(\)](#) ([bitshares.aio.genesisbalance.GenesisBalances method](#)), 58

[index\(\)](#) ([bitshares.aio.proposal.Proposals method](#)), 81

[index\(\)](#) ([bitshares.aio.witness.Witnesses method](#)), 92

[index\(\)](#) ([bitshares.aio.worker.Workers method](#)), 96

[index\(\)](#) ([bitshares.genesisbalance.GenesisBalances method](#)), 138

[index\(\)](#) ([bitshares.proposal.Proposals method](#)), 163

[index\(\)](#) ([bitshares.witness.Witnesses method](#)), 174

[index\(\)](#) ([bitshares.worker.Workers method](#)), 178

[index\(\)](#) ([bitsharesbase.objects.Operation method](#)), 188

[info\(\)](#) ([bitshares.aio.bitshares.BitShares method](#)), 37

[info\(\)](#) ([bitshares.aio.blockchain.Blockchain method](#)), 46

[info\(\)](#) ([bitshares.bitshares.BitShares method](#)), 116

[info\(\)](#) ([bitshares.blockchain.Blockchain method](#)), 125

[inject\(\)](#) ([bitshares.account.Account class method](#)), 99

[inject\(\)](#) ([bitshares.account.AccountUpdate class method](#)), 101

[inject\(\)](#) ([bitshares.aio.account.Account class method](#)), 21

[inject\(\)](#) ([bitshares.aio.account.AccountUpdate class method](#)), 23

[inject\(\)](#) ([bitshares.aio.amount.Amount class method](#)), 25

[inject\(\)](#) ([bitshares.aio.asset.Asset class method](#)), 27

[inject\(\)](#) ([bitshares.aio.block.Block class method](#)), 42

[inject\(\)](#) ([bitshares.aio.block.BlockHeader class method](#)), 43

[inject\(\)](#) ([bitshares.aio.blockchain.Blockchain class method](#)), 46

[inject\(\)](#) ([bitshares.aio.blockchainobject.BlockchainObject class method](#)), 48

[inject\(\)](#) ([bitshares.aio.blockchainobject.Object class method](#)), 50

[inject\(\)](#) ([bitshares.aio.committee.Committee class method](#)), 52

[inject\(\)](#) ([bitshares.aio.dex.Dex class method](#)), 55

[inject\(\)](#) ([bitshares.aio.genesisbalance.GenesisBalance class method](#)), 56

[inject\(\)](#) ([bitshares.aio.genesisbalance.GenesisBalances class method](#)), 58

[inject\(\)](#) ([bitshares.aio.htlc.Htlc class method](#)), 60

[inject\(\)](#) ([bitshares.aio.instance.BlockchainInstance class method](#)), 61

[inject\(\)](#) ([bitshares.aio.market.Market class method](#)), 64

[inject\(\)](#) ([bitshares.aio.memo.Memo class method](#)), 68

[inject\(\)](#) ([bitshares.aio.message.Message class method](#)), 69

[inject\(\)](#) ([bitshares.aio.price.FilledOrder class method](#)), 70

[inject\(\)](#) ([bitshares.aio.price.Order class method](#)), 72

[inject\(\)](#) ([bitshares.aio.price.Price class method](#)), 74

[inject\(\)](#) ([bitshares.aio.price.PriceFeed class method](#)), 76

[inject\(\)](#) ([bitshares.aio.price.UpdateCallOrder class method](#)), 77

[inject\(\)](#) ([bitshares.aio.proposal.Proposal class method](#)), 79

[inject\(\)](#) ([bitshares.aio.proposal.Proposals class method](#)), 81

[inject\(\)](#) ([bitshares.aio.transactionbuilder.ProposalBuilder class method](#)), 83

[inject\(\)](#) ([bitshares.aio.transactionbuilder.TransactionBuilder class method](#)), 84

[inject\(\)](#) ([bitshares.aio.vesting.Vesting class method](#)), 86

[inject\(\)](#) ([bitshares.aio.wallet.Wallet class method](#)), 89

[inject\(\)](#) ([bitshares.aio.witness.Witness class method](#)), 90

- `inject()` (*bitshares.aio.witness.Witnesses class method*), 93
- `inject()` (*bitshares.aio.worker.Worker class method*), 94
- `inject()` (*bitshares.aio.worker.Workers class method*), 96
- `inject()` (*bitshares.amount.Amount class method*), 103
- `inject()` (*bitshares.asset.Asset class method*), 106
- `inject()` (*bitshares.block.Block class method*), 121
- `inject()` (*bitshares.block.BlockHeader class method*), 122
- `inject()` (*bitshares.blockchain.Blockchain class method*), 125
- `inject()` (*bitshares.blockchainobject.BlockchainObject class method*), 127
- `inject()` (*bitshares.blockchainobject.Object class method*), 129
- `inject()` (*bitshares.committee.Committee class method*), 131
- `inject()` (*bitshares.dex.Dex class method*), 134
- `inject()` (*bitshares.genesisbalance.GenesisBalance class method*), 136
- `inject()` (*bitshares.genesisbalance.GenesisBalances class method*), 138
- `inject()` (*bitshares.htlc.Htlc class method*), 139
- `inject()` (*bitshares.instance.BlockchainInstance class method*), 141
- `inject()` (*bitshares.market.Market class method*), 144
- `inject()` (*bitshares.memo.Memo class method*), 148
- `inject()` (*bitshares.message.Message class method*), 148
- `inject()` (*bitshares.notify.Notify class method*), 150
- `inject()` (*bitshares.price.FilledOrder class method*), 152
- `inject()` (*bitshares.price.Order class method*), 153
- `inject()` (*bitshares.price.Price class method*), 156
- `inject()` (*bitshares.price.PriceFeed class method*), 157
- `inject()` (*bitshares.price.UpdateCallOrder class method*), 159
- `inject()` (*bitshares.proposal.Proposal class method*), 160
- `inject()` (*bitshares.proposal.Proposals class method*), 163
- `inject()` (*bitshares.transactionbuilder.ProposalBuilder class method*), 164
- `inject()` (*bitshares.transactionbuilder.TransactionBuilder class method*), 166
- `inject()` (*bitshares.vesting.Vesting class method*), 168
- `inject()` (*bitshares.wallet.Wallet class method*), 170
- `inject()` (*bitshares.witness.Witness class method*), 172
- `inject()` (*bitshares.witness.Witnesses class method*), 174
- `inject()` (*bitshares.worker.Worker class method*), 176
- `inject()` (*bitshares.worker.Workers class method*), 178
- `insert()` (*bitshares.aio.genesisbalance.GenesisBalances method*), 58
- `insert()` (*bitshares.aio.proposal.Proposals method*), 81
- `insert()` (*bitshares.aio.witness.Witnesses method*), 93
- `insert()` (*bitshares.aio.worker.Workers method*), 96
- `insert()` (*bitshares.genesisbalance.GenesisBalances method*), 138
- `insert()` (*bitshares.proposal.Proposals method*), 163
- `insert()` (*bitshares.witness.Witnesses method*), 174
- `insert()` (*bitshares.worker.Workers method*), 178
- `insert()` (*bitsharesbase.objects.Operation method*), 188
- `instance` (*bitshares.aio.instance.SharedInstance attribute*), 61
- `instance` (*bitshares.instance.SharedInstance attribute*), 141
- `invert()` (*bitshares.aio.price.FilledOrder method*), 70
- `invert()` (*bitshares.aio.price.Order method*), 72
- `invert()` (*bitshares.aio.price.Price method*), 74
- `invert()` (*bitshares.aio.price.UpdateCallOrder method*), 77
- `invert()` (*bitshares.price.FilledOrder method*), 152
- `invert()` (*bitshares.price.Order method*), 153
- `invert()` (*bitshares.price.Price method*), 156
- `invert()` (*bitshares.price.UpdateCallOrder method*), 159
- `is_active` (*bitshares.aio.witness.Witness attribute*), 90
- `is_active` (*bitshares.witness.Witness attribute*), 172
- `is_bitasset` (*bitshares.aio.asset.Asset attribute*), 27
- `is_bitasset` (*bitshares.asset.Asset attribute*), 106
- `is_connected()` (*bitshares.aio.bitshares.BitShares method*), 37
- `is_connected()` (*bitshares.bitshares.BitShares method*), 116
- `is_empty()` (*bitshares.aio.transactionbuilder.ProposalBuilder method*), 83
- `is_empty()` (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 84
- `is_empty()` (*bitshares.transactionbuilder.ProposalBuilder method*), 164
- `is_empty()` (*bitshares.transactionbuilder.TransactionBuilder method*), 166
- `is_encrypted()` (*bitshares.aio.wallet.Wallet method*), 89
- `is_encrypted()` (*bitshares.wallet.Wallet method*), 170
- `is_fully_loaded` (*bitshares.account.Account attribute*), 99
- `is_fully_loaded` (*bitshares.aio.account.Account attribute*), 99

tribute), 21

is\_fully\_loaded (bitshares.aio.asset.Asset attribute), 27

is\_fully\_loaded (bitshares.asset.Asset attribute), 106

is\_in\_review (bitshares.aio.proposal.Proposal attribute), 79

is\_in\_review (bitshares.proposal.Proposal attribute), 160

is\_irreversible\_mode() (bitshares.aio.blockchain.Blockchain method), 46

is\_irreversible\_mode() (bitshares.blockchain.Blockchain method), 125

is\_ltm (bitshares.account.Account attribute), 99

is\_ltm (bitshares.aio.account.Account attribute), 21

issue() (bitshares.aio.asset.Asset method), 27

issue() (bitshares.asset.Asset method), 106

items() (bitshares.account.Account method), 99

items() (bitshares.account.AccountUpdate method), 101

items() (bitshares.aio.account.Account method), 21

items() (bitshares.aio.account.AccountUpdate method), 23

items() (bitshares.aio.amount.Amount method), 25

items() (bitshares.aio.asset.Asset method), 27

items() (bitshares.aio.block.Block method), 42

items() (bitshares.aio.block.BlockHeader method), 43

items() (bitshares.aio.blockchainobject.BlockchainObject method), 48

items() (bitshares.aio.blockchainobject.Object method), 50

items() (bitshares.aio.committee.Committee method), 52

items() (bitshares.aio.genesisbalance.GenesisBalance method), 56

items() (bitshares.aio.htlc.Htlc method), 60

items() (bitshares.aio.market.Market method), 64

items() (bitshares.aio.price.FilledOrder method), 70

items() (bitshares.aio.price.Order method), 72

items() (bitshares.aio.price.Price method), 74

items() (bitshares.aio.price.PriceFeed method), 76

items() (bitshares.aio.price.UpdateCallOrder method), 77

items() (bitshares.aio.proposal.Proposal method), 79

items() (bitshares.aio.proposal.Proposals method), 81

items() (bitshares.aio.transactionbuilder.TransactionBuilder method), 85

items() (bitshares.aio.vesting.Vesting method), 86

items() (bitshares.aio.witness.Witness method), 90

items() (bitshares.aio.witness.Witnesses method), 93

items() (bitshares.aio.worker.Worker method), 94

items() (bitshares.aio.worker.Workers method), 96

items() (bitshares.amount.Amount method), 103

items() (bitshares.asset.Asset method), 106

items() (bitshares.block.Block method), 121

items() (bitshares.block.BlockHeader method), 122

items() (bitshares.blockchainobject.BlockchainObject method), 127

items() (bitshares.blockchainobject.Object method), 129

items() (bitshares.committee.Committee method), 131

items() (bitshares.genesisbalance.GenesisBalance method), 136

items() (bitshares.htlc.Htlc method), 139

items() (bitshares.market.Market method), 144

items() (bitshares.price.FilledOrder method), 152

items() (bitshares.price.Order method), 153

items() (bitshares.price.Price method), 156

items() (bitshares.price.PriceFeed method), 157

items() (bitshares.price.UpdateCallOrder method), 159

items() (bitshares.proposal.Proposal method), 160

items() (bitshares.proposal.Proposals method), 163

items() (bitshares.transactionbuilder.TransactionBuilder method), 166

items() (bitshares.vesting.Vesting method), 168

items() (bitshares.witness.Witness method), 172

items() (bitshares.witness.Witnesses method), 174

items() (bitshares.worker.Worker method), 176

items() (bitshares.worker.Workers method), 178

items() (bitsharesbase.objects.AccountCreateExtensions.Buyback\_option method), 183

items() (bitsharesbase.objects.AccountCreateExtensions.Null\_ext method), 184

items() (bitsharesbase.objects.AccountOptions method), 185

items() (bitsharesbase.objects.AssetOptions method), 185

items() (bitsharesbase.objects.BitAssetOptions method), 186

items() (bitsharesbase.objects.Memo method), 187

items() (bitsharesbase.objects.Permission method), 189

items() (bitsharesbase.objects.Price method), 190

items() (bitsharesbase.objects.PriceFeed method), 190

items() (bitsharesbase.operations.Account\_create method), 192

items() (bitsharesbase.operations.Account\_update method), 192

items() (bitsharesbase.operations.Account\_upgrade method), 193

items() (bitsharesbase.operations.Account\_whitelist method), 194

items() (bitsharesbase.operations.Assert method), 195

items() (bitsharesbase.operations.Asset\_claim\_fees method), 195



- `items()` (`bitsharesbase.operations.Asset_claim_pool` method), 196
  - `items()` (`bitsharesbase.operations.Asset_create` method), 197
  - `items()` (`bitsharesbase.operations.Asset_fund_fee_pool` method), 197
  - `items()` (`bitsharesbase.operations.Asset_global_settle` method), 198
  - `items()` (`bitsharesbase.operations.Asset_issue` method), 199
  - `items()` (`bitsharesbase.operations.Asset_publish_feed` method), 200
  - `items()` (`bitsharesbase.operations.Asset_reserve` method), 200
  - `items()` (`bitsharesbase.operations.Asset_settle` method), 201
  - `items()` (`bitsharesbase.operations.Asset_update` method), 202
  - `items()` (`bitsharesbase.operations.Asset_update_bitasset` method), 202
  - `items()` (`bitsharesbase.operations.Asset_update_feed_producers` method), 203
  - `items()` (`bitsharesbase.operations.Asset_update_issuer` method), 204
  - `items()` (`bitsharesbase.operations.Balance_claim` method), 205
  - `items()` (`bitsharesbase.operations.Bid_collateral` method), 205
  - `items()` (`bitsharesbase.operations.Call_order_update` method), 206
  - `items()` (`bitsharesbase.operations.Committee_member_create` method), 207
  - `items()` (`bitsharesbase.operations.Custom` method), 207
  - `items()` (`bitsharesbase.operations.Htlc_create` method), 208
  - `items()` (`bitsharesbase.operations.Htlc_extend` method), 209
  - `items()` (`bitsharesbase.operations.Htlc_redeem` method), 210
  - `items()` (`bitsharesbase.operations.Limit_order_cancel` method), 210
  - `items()` (`bitsharesbase.operations.Limit_order_create` method), 211
  - `items()` (`bitsharesbase.operations.Liquidity_pool_create` method), 212
  - `items()` (`bitsharesbase.operations.Liquidity_pool_delete` method), 213
  - `items()` (`bitsharesbase.operations.Liquidity_pool_deposit` method), 213
  - `items()` (`bitsharesbase.operations.Liquidity_pool_exchange` method), 214
  - `items()` (`bitsharesbase.operations.Liquidity_pool_withdraw` method), 215
  - `items()` (`bitsharesbase.operations.Op_wrapper` method), 215
  - `items()` (`bitsharesbase.operations.Override_transfer` method), 216
  - `items()` (`bitsharesbase.operations.Proposal_create` method), 217
  - `items()` (`bitsharesbase.operations.Proposal_update` method), 218
  - `items()` (`bitsharesbase.operations.Ticket_create_operation` method), 218
  - `items()` (`bitsharesbase.operations.Ticket_update_operation` method), 219
  - `items()` (`bitsharesbase.operations.Transfer` method), 220
  - `items()` (`bitsharesbase.operations.Vesting_balance_withdraw` method), 220
  - `items()` (`bitsharesbase.operations.Withdraw_permission_create` method), 221
  - `items()` (`bitsharesbase.operations.Witness_update` method), 222
  - `items()` (`bitsharesbase.operations.Worker_create` method), 223
  - `items()` (`bitsharesbase.signedtransactions.Signed_Transaction` method), 224
- ## J
- `json()` (`bitshares.aio.amount.Amount` method), 25
  - `json()` (`bitshares.aio.price.FilledOrder` method), 70
  - `json()` (`bitshares.aio.price.Order` method), 72
  - `json()` (`bitshares.aio.price.Price` method), 74
  - `json()` (`bitshares.aio.price.UpdateCallOrder` method), 77
  - `json()` (`bitshares.aio.transactionbuilder.ProposalBuilder` method), 83
  - `json()` (`bitshares.aio.transactionbuilder.TransactionBuilder` method), 85
  - `json()` (`bitshares.amount.Amount` method), 103
  - `json()` (`bitshares.price.FilledOrder` method), 152
  - `json()` (`bitshares.price.Order` method), 153
  - `json()` (`bitshares.price.Price` method), 156
  - `json()` (`bitshares.price.UpdateCallOrder` method), 159
  - `json()` (`bitshares.transactionbuilder.ProposalBuilder` method), 164
  - `json()` (`bitshares.transactionbuilder.TransactionBuilder` method), 166
  - `json()` (`bitsharesbase.objects.AccountCreateExtensions.Buyback_options` method), 183
  - `json()` (`bitsharesbase.objects.AccountCreateExtensions.Null_ext` method), 184
  - `json()` (`bitsharesbase.objects.AccountOptions` method), 185
  - `json()` (`bitsharesbase.objects.AssetOptions` method), 185

`json()` (*bitsharesbase.objects.BitAssetOptions method*), 186

`json()` (*bitsharesbase.objects.Memo method*), 187

`json()` (*bitsharesbase.objects.Operation method*), 188

`json()` (*bitsharesbase.objects.Permission method*), 189

`json()` (*bitsharesbase.objects.Price method*), 190

`json()` (*bitsharesbase.objects.PriceFeed method*), 190

`json()` (*bitsharesbase.operations.Account\_create method*), 192

`json()` (*bitsharesbase.operations.Account\_update method*), 192

`json()` (*bitsharesbase.operations.Account\_upgrade method*), 193

`json()` (*bitsharesbase.operations.Account\_whitelist method*), 194

`json()` (*bitsharesbase.operations.Assert method*), 195

`json()` (*bitsharesbase.operations.Asset\_claim\_fees method*), 195

`json()` (*bitsharesbase.operations.Asset\_claim\_pool method*), 196

`json()` (*bitsharesbase.operations.Asset\_create method*), 197

`json()` (*bitsharesbase.operations.Asset\_fund\_fee\_pool method*), 197

`json()` (*bitsharesbase.operations.Asset\_global\_settle method*), 198

`json()` (*bitsharesbase.operations.Asset\_issue method*), 199

`json()` (*bitsharesbase.operations.Asset\_publish\_feed method*), 200

`json()` (*bitsharesbase.operations.Asset\_reserve method*), 200

`json()` (*bitsharesbase.operations.Asset\_settle method*), 201

`json()` (*bitsharesbase.operations.Asset\_update method*), 202

`json()` (*bitsharesbase.operations.Asset\_update\_bitasset method*), 202

`json()` (*bitsharesbase.operations.Asset\_update\_feed\_producers method*), 203

`json()` (*bitsharesbase.operations.Asset\_update\_issuer method*), 204

`json()` (*bitsharesbase.operations.Balance\_claim method*), 205

`json()` (*bitsharesbase.operations.Bid\_collateral method*), 205

`json()` (*bitsharesbase.operations.Call\_order\_update method*), 206

`json()` (*bitsharesbase.operations.Committee\_member\_create method*), 207

`json()` (*bitsharesbase.operations.Custom method*), 207

`json()` (*bitsharesbase.operations.Htlc\_create method*), 208

`json()` (*bitsharesbase.operations.Htlc\_extend method*), 209

`json()` (*bitsharesbase.operations.Htlc\_redeem method*), 210

`json()` (*bitsharesbase.operations.Limit\_order\_cancel method*), 210

`json()` (*bitsharesbase.operations.Limit\_order\_create method*), 211

`json()` (*bitsharesbase.operations.Liquidity\_pool\_create method*), 212

`json()` (*bitsharesbase.operations.Liquidity\_pool\_delete method*), 213

`json()` (*bitsharesbase.operations.Liquidity\_pool\_deposit method*), 213

`json()` (*bitsharesbase.operations.Liquidity\_pool\_exchange method*), 214

`json()` (*bitsharesbase.operations.Liquidity\_pool\_withdraw method*), 215

`json()` (*bitsharesbase.operations.Op\_wrapper method*), 215

`json()` (*bitsharesbase.operations.Override\_transfer method*), 216

`json()` (*bitsharesbase.operations.Proposal\_create method*), 217

`json()` (*bitsharesbase.operations.Proposal\_update method*), 218

`json()` (*bitsharesbase.operations.Ticket\_create\_operation method*), 218

`json()` (*bitsharesbase.operations.Ticket\_update\_operation method*), 219

`json()` (*bitsharesbase.operations.Transfer method*), 220

`json()` (*bitsharesbase.operations.Vesting\_balance\_withdraw method*), 220

`json()` (*bitsharesbase.operations.Withdraw\_permission\_create method*), 221

`json()` (*bitsharesbase.operations.Witness\_update method*), 222

`json()` (*bitsharesbase.operations.Worker\_create method*), 223

`json()` (*bitsharesbase.signedtransactions.Signed\_Transaction method*), 224

## K

`keys()` (*bitshares.account.Account method*), 99

`keys()` (*bitshares.account.AccountUpdate method*), 101

`keys()` (*bitshares.aio.account.Account method*), 21

`keys()` (*bitshares.aio.account.AccountUpdate method*), 23

`keys()` (*bitshares.aio.amount.Amount method*), 25

`keys()` (*bitshares.aio.asset.Asset method*), 27

`keys()` (*bitshares.aio.block.Block method*), 42

`keys()` (*bitshares.aio.block.BlockHeader method*), 44

- keys () (*bitshares.aio.blockchainobject.BlockchainObject* method), 48
- keys () (*bitshares.aio.blockchainobject.Object* method), 50
- keys () (*bitshares.aio.committee.Committee* method), 52
- keys () (*bitshares.aio.genesisbalance.GenesisBalance* method), 56
- keys () (*bitshares.aio.htlc.Htlc* method), 60
- keys () (*bitshares.aio.market.Market* method), 64
- keys () (*bitshares.aio.price.FilledOrder* method), 70
- keys () (*bitshares.aio.price.Order* method), 72
- keys () (*bitshares.aio.price.Price* method), 74
- keys () (*bitshares.aio.price.PriceFeed* method), 76
- keys () (*bitshares.aio.price.UpdateCallOrder* method), 77
- keys () (*bitshares.aio.proposal.Proposal* method), 79
- keys () (*bitshares.aio.transactionbuilder.TransactionBuilder* method), 85
- keys () (*bitshares.aio.vesting.Vesting* method), 87
- keys () (*bitshares.aio.witness.Witness* method), 90
- keys () (*bitshares.aio.worker.Worker* method), 94
- keys () (*bitshares.amount.Amount* method), 103
- keys () (*bitshares.asset.Asset* method), 106
- keys () (*bitshares.block.Block* method), 121
- keys () (*bitshares.block.BlockHeader* method), 122
- keys () (*bitshares.blockchainobject.BlockchainObject* method), 127
- keys () (*bitshares.blockchainobject.Object* method), 129
- keys () (*bitshares.committee.Committee* method), 131
- keys () (*bitshares.genesisbalance.GenesisBalance* method), 136
- keys () (*bitshares.htlc.Htlc* method), 139
- keys () (*bitshares.market.Market* method), 144
- keys () (*bitshares.price.FilledOrder* method), 152
- keys () (*bitshares.price.Order* method), 153
- keys () (*bitshares.price.Price* method), 156
- keys () (*bitshares.price.PriceFeed* method), 157
- keys () (*bitshares.price.UpdateCallOrder* method), 159
- keys () (*bitshares.proposal.Proposal* method), 160
- keys () (*bitshares.transactionbuilder.TransactionBuilder* method), 166
- keys () (*bitshares.vesting.Vesting* method), 168
- keys () (*bitshares.witness.Witness* method), 172
- keys () (*bitshares.worker.Worker* method), 176
- keys () (*bitsharesbase.objects.AccountCreateExtensions.Buyback\_order* method), 183
- keys () (*bitsharesbase.objects.AccountCreateExtensions.Null\_ext* method), 184
- keys () (*bitsharesbase.objects.AccountOptions* method), 185
- keys () (*bitsharesbase.objects.AssetOptions* method), 185
- keys () (*bitsharesbase.objects.BitAssetOptions* method), 186
- keys () (*bitsharesbase.objects.Memo* method), 187
- keys () (*bitsharesbase.objects.Permission* method), 189
- keys () (*bitsharesbase.objects.Price* method), 190
- keys () (*bitsharesbase.objects.PriceFeed* method), 190
- keys () (*bitsharesbase.operations.Account\_create* method), 192
- keys () (*bitsharesbase.operations.Account\_update* method), 192
- keys () (*bitsharesbase.operations.Account\_upgrade* method), 193
- keys () (*bitsharesbase.operations.Account\_whitelist* method), 194
- keys () (*bitsharesbase.operations.Assert* method), 195
- keys () (*bitsharesbase.operations.Asset\_claim\_fees* method), 195
- keys () (*bitsharesbase.operations.Asset\_claim\_pool* method), 196
- keys () (*bitsharesbase.operations.Asset\_create* method), 197
- keys () (*bitsharesbase.operations.Asset\_fund\_fee\_pool* method), 197
- keys () (*bitsharesbase.operations.Asset\_global\_settle* method), 198
- keys () (*bitsharesbase.operations.Asset\_issue* method), 199
- keys () (*bitsharesbase.operations.Asset\_publish\_feed* method), 200
- keys () (*bitsharesbase.operations.Asset\_reserve* method), 200
- keys () (*bitsharesbase.operations.Asset\_settle* method), 201
- keys () (*bitsharesbase.operations.Asset\_update* method), 202
- keys () (*bitsharesbase.operations.Asset\_update\_bitasset* method), 202
- keys () (*bitsharesbase.operations.Asset\_update\_feed\_producers* method), 203
- keys () (*bitsharesbase.operations.Asset\_update\_issuer* method), 204
- keys () (*bitsharesbase.operations.Balance\_claim* method), 205
- keys () (*bitsharesbase.operations.Bid\_collateral* method), 205
- keys () (*bitsharesbase.operations.Call\_order\_update* method), 206
- keys () (*bitsharesbase.operations.Committee\_member\_create* method), 207
- keys () (*bitsharesbase.operations.Custom* method), 208
- keys () (*bitsharesbase.operations.Htlc\_create* method), 208
- keys () (*bitsharesbase.operations.Htlc\_extend* method), 209

- keys () (*bitsharesbase.operations.Htlc\_redeem method*), 210  
 keys () (*bitsharesbase.operations.Limit\_order\_cancel method*), 211  
 keys () (*bitsharesbase.operations.Limit\_order\_create method*), 211  
 keys () (*bitsharesbase.operations.Liquidity\_pool\_create method*), 212  
 keys () (*bitsharesbase.operations.Liquidity\_pool\_delete method*), 213  
 keys () (*bitsharesbase.operations.Liquidity\_pool\_deposit method*), 213  
 keys () (*bitsharesbase.operations.Liquidity\_pool\_exchange method*), 214  
 keys () (*bitsharesbase.operations.Liquidity\_pool\_withdraw method*), 215  
 keys () (*bitsharesbase.operations.Op\_wrapper method*), 216  
 keys () (*bitsharesbase.operations.Override\_transfer method*), 216  
 keys () (*bitsharesbase.operations.Proposal\_create method*), 217  
 keys () (*bitsharesbase.operations.Proposal\_update method*), 218  
 keys () (*bitsharesbase.operations.Ticket\_create\_operation method*), 218  
 keys () (*bitsharesbase.operations.Ticket\_update\_operation method*), 219  
 keys () (*bitsharesbase.operations.Transfer method*), 220  
 keys () (*bitsharesbase.operations.Vesting\_balance\_withdraw method*), 221  
 keys () (*bitsharesbase.operations.Withdraw\_permission\_create method*), 221  
 keys () (*bitsharesbase.operations.Witness\_update method*), 222  
 keys () (*bitsharesbase.operations.Worker\_create method*), 223  
 keys () (*bitsharesbase.signedtransactions.Signed\_Transaction method*), 224  
 klass () (*bitsharesbase.objects.Operation method*), 188  
 klass\_name (*bitsharesbase.objects.Operation attribute*), 188  
 known\_chains (*bitsharesbase.signedtransactions.Signed\_Transaction attribute*), 224
- L**
- Limit\_order\_cancel (*class in bitsharesbase.operations*), 210  
 Limit\_order\_create (*class in bitsharesbase.operations*), 211  
 Liquidity\_pool\_create (*class in bitsharesbase.operations*), 212  
 Liquidity\_pool\_delete (*class in bitsharesbase.operations*), 212  
 Liquidity\_pool\_deposit (*class in bitsharesbase.operations*), 213  
 Liquidity\_pool\_exchange (*class in bitsharesbase.operations*), 214  
 Liquidity\_pool\_withdraw (*class in bitsharesbase.operations*), 215  
 list\_debt\_positions () (*bitshares.aio.dex.Dex method*), 55  
 list\_debt\_positions () (*bitshares.dex.Dex method*), 134  
 list\_operations () (*bitshares.aio.transactionbuilder.ProposalBuilder method*), 83  
 list\_operations () (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 85  
 list\_operations () (*bitshares.transactionbuilder.ProposalBuilder method*), 164  
 list\_operations () (*bitshares.transactionbuilder.TransactionBuilder method*), 166  
 listen () (*bitshares.notify.Notify method*), 150  
 lock () (*bitshares.aio.wallet.Wallet method*), 89  
 lock () (*bitshares.wallet.Wallet method*), 170  
 locked () (*bitshares.aio.wallet.Wallet method*), 89  
 locked () (*bitshares.wallet.Wallet method*), 170
- M**
- market (*bitshares.aio.price.FilledOrder attribute*), 70  
 market (*bitshares.aio.price.Order attribute*), 72  
 market (*bitshares.aio.price.Price attribute*), 75  
 market (*bitshares.aio.price.UpdateCallOrder attribute*), 77  
 market (*bitshares.price.FilledOrder attribute*), 152  
 market (*bitshares.price.Order attribute*), 153  
 market (*bitshares.price.Price attribute*), 156  
 market (*bitshares.price.UpdateCallOrder attribute*), 159  
 Market (*class in bitshares.aio.market*), 62  
 Market (*class in bitshares.market*), 141  
 market\_fee\_percent (*bitshares.aio.asset.Asset attribute*), 27  
 market\_fee\_percent (*bitshares.asset.Asset attribute*), 106  
 max\_market\_fee (*bitshares.aio.asset.Asset attribute*), 28  
 max\_market\_fee (*bitshares.asset.Asset attribute*), 106  
 Memo (*class in bitshares.aio.memo*), 67

- Memo (class in *bitshares.memo*), 146
- Memo (class in *bitsharesbase.objects*), 187
- Message (class in *bitshares.aio.message*), 68
- Message (class in *bitshares.message*), 148
- MESSAGE\_SPLIT (*bitshares.aio.message.Message* attribute), 68
- MESSAGE\_SPLIT (*bitshares.message.Message* attribute), 148
- module (*bitsharesbase.objects.Operation* attribute), 188
- move\_to\_end() (*bitsharesbase.objects.AccountCreateExtensions.Buyback\_options* method), 183
- move\_to\_end() (*bitsharesbase.objects.AccountCreateExtensions.Null\_ext* method), 184
- move\_to\_end() (*bitsharesbase.objects.AccountOptions* method), 185
- move\_to\_end() (*bitsharesbase.objects.AssetOptions* method), 185
- move\_to\_end() (*bitsharesbase.objects.BitAssetOptions* method), 186
- move\_to\_end() (*bitsharesbase.objects.Memo* method), 187
- move\_to\_end() (*bitsharesbase.objects.Permission* method), 189
- move\_to\_end() (*bitsharesbase.objects.Price* method), 190
- move\_to\_end() (*bitsharesbase.objects.PriceFeed* method), 190
- move\_to\_end() (*bitsharesbase.operations.Account\_create* method), 192
- move\_to\_end() (*bitsharesbase.operations.Account\_update* method), 192
- move\_to\_end() (*bitsharesbase.operations.Account\_upgrade* method), 193
- move\_to\_end() (*bitsharesbase.operations.Account\_whitelist* method), 194
- move\_to\_end() (*bitsharesbase.operations.Assert* method), 195
- move\_to\_end() (*bitsharesbase.operations.Asset\_claim\_fees* method), 195
- move\_to\_end() (*bitsharesbase.operations.Asset\_claim\_pool* method), 196
- move\_to\_end() (*bitsharesbase.operations.Asset\_create* method), 197
- move\_to\_end() (*bitsharesbase.operations.Asset\_fund\_fee\_pool* method), 198
- move\_to\_end() (*bitsharesbase.operations.Asset\_global\_settle* method), 198
- move\_to\_end() (*bitsharesbase.operations.Asset\_issue* method), 199
- move\_to\_end() (*bitsharesbase.operations.Asset\_publish\_feed* method), 200
- move\_to\_end() (*bitsharesbase.operations.Asset\_reserve* method), 200
- move\_to\_end() (*bitsharesbase.operations.Asset\_settle* method), 201
- move\_to\_end() (*bitsharesbase.operations.Asset\_update* method), 202
- move\_to\_end() (*bitsharesbase.operations.Asset\_update\_bitasset* method), 203
- move\_to\_end() (*bitsharesbase.operations.Asset\_update\_feed\_producers* method), 203
- move\_to\_end() (*bitsharesbase.operations.Asset\_update\_issuer* method), 204
- move\_to\_end() (*bitsharesbase.operations.Balance\_claim* method), 205
- move\_to\_end() (*bitsharesbase.operations.Bid\_collateral* method), 205
- move\_to\_end() (*bitsharesbase.operations.Call\_order\_update* method), 206
- move\_to\_end() (*bitsharesbase.operations.Committee\_member\_create* method), 207
- move\_to\_end() (*bitsharesbase.operations.Custom* method), 208
- move\_to\_end() (*bitsharesbase.operations.Htlc\_create* method), 208
- move\_to\_end() (*bitsharesbase.operations.Htlc\_extend* method), 209
- move\_to\_end() (*bitsharesbase.operations.Htlc\_redeem* method), 210
- move\_to\_end() (*bitsharesbase.operations.Limit\_order\_cancel* method), 211
- move\_to\_end() (*bitsharesbase.operations.Limit\_order\_create* method), 211
- move\_to\_end() (*bitsharesbase.operations.Liquidity\_pool\_create* method), 212
- move\_to\_end() (*bitshares-*



*base.operations.Liquidity\_pool\_delete method*), 213

*move\_to\_end()* (*bitshares-base.operations.Liquidity\_pool\_deposit method*), 213

*move\_to\_end()* (*bitshares-base.operations.Liquidity\_pool\_exchange method*), 214

*move\_to\_end()* (*bitshares-base.operations.Liquidity\_pool\_withdraw method*), 215

*move\_to\_end()* (*bitshares-base.operations.Op\_wrapper method*), 216

*move\_to\_end()* (*bitshares-base.operations.Override\_transfer method*), 216

*move\_to\_end()* (*bitshares-base.operations.Proposal\_create method*), 217

*move\_to\_end()* (*bitshares-base.operations.Proposal\_update method*), 218

*move\_to\_end()* (*bitshares-base.operations.Ticket\_create\_operation method*), 218

*move\_to\_end()* (*bitshares-base.operations.Ticket\_update\_operation method*), 219

*move\_to\_end()* (*bitsharesbase.operations.Transfer method*), 220

*move\_to\_end()* (*bitshares-base.operations.Vesting\_balance\_withdraw method*), 221

*move\_to\_end()* (*bitshares-base.operations.Withdraw\_permission\_create method*), 221

*move\_to\_end()* (*bitshares-base.operations.Witness\_update method*), 222

*move\_to\_end()* (*bitshares-base.operations.Worker\_create method*), 223

*move\_to\_end()* (*bitshares-base.signedtransactions.Signed\_Transaction method*), 224

## N

*name* (*bitshares.account.Account attribute*), 99

*name* (*bitshares.aio.account.Account attribute*), 21

*new\_proposal()* (*bitshares.aio.bitshares.BitShares method*), 37

*new\_proposal()* (*bitshares.bitshares.BitShares method*), 116

*new\_tx()* (*bitshares.aio.bitshares.BitShares method*), 37

*new\_tx()* (*bitshares.bitshares.BitShares method*), 117

*new\_wallet()* (*bitshares.aio.bitshares.BitShares method*), 37

*new\_wallet()* (*bitshares.bitshares.BitShares method*), 117

*newWallet()* (*bitshares.aio.bitshares.BitShares method*), 37

*newWallet()* (*bitshares.aio.wallet.Wallet method*), 89

*newWallet()* (*bitshares.bitshares.BitShares method*), 116

*newWallet()* (*bitshares.wallet.Wallet method*), 170

*next\_sequence()* (*bitsharesbase.account.BrainKey method*), 180

*no\_listing* (*bitshares-base.operations.Account\_whitelist attribute*), 194

*nolist()* (*bitshares.account.Account method*), 99

*nolist()* (*bitshares.aio.account.Account method*), 21

*normalize()* (*bitsharesbase.account.BrainKey method*), 180

*Notify* (*class in bitshares.notify*), 149

## O

*Object* (*class in bitshares.aio.blockchainobject*), 49

*Object* (*class in bitshares.blockchainobject*), 128

*object\_type* (*in module bitsharesbase.objecttypes*), 191

*object\_types* (*bitsharesbase.objects.ObjectId attribute*), 188

*ObjectId* (*class in bitsharesbase.objects*), 187

*objectid\_valid()* (*bitshares.account.Account static method*), 99

*objectid\_valid()* (*bitshares.aio.account.Account static method*), 21

*objectid\_valid()* (*bitshares.aio.asset.Asset static method*), 28

*objectid\_valid()* (*bitshares.aio.block.Block static method*), 42

*objectid\_valid()* (*bitshares.aio.block.BlockHeader static method*), 44

*objectid\_valid()* (*bitshares.aio.blockchainobject.BlockchainObject static method*), 48

*objectid\_valid()* (*bitshares.aio.blockchainobject.Object static method*), 50

*objectid\_valid()* (*bitshares.aio.committee.Committee static method*), 52

*objectid\_valid()* (*bitshares.aio.genesisbalance.GenesisBalance static method*), 52

static method), 56  
 objectid\_valid() (bitshares.aio.htlc.Htlc static method), 60  
 objectid\_valid() (bitshares.aio.proposal.Proposal static method), 79  
 objectid\_valid() (bitshares.aio.vesting.Vesting static method), 87  
 objectid\_valid() (bitshares.aio.witness.Witness static method), 91  
 objectid\_valid() (bitshares.aio.worker.Worker static method), 94  
 objectid\_valid() (bitshares.asset.Asset static method), 106  
 objectid\_valid() (bitshares.block.Block static method), 121  
 objectid\_valid() (bitshares.block.BlockHeader static method), 123  
 objectid\_valid() (bitshares.blockchainobject.BlockchainObject static method), 127  
 objectid\_valid() (bitshares.blockchainobject.Object static method), 129  
 objectid\_valid() (bitshares.committee.Committee static method), 131  
 objectid\_valid() (bitshares.genesisbalance.GenesisBalance static method), 136  
 objectid\_valid() (bitshares.htlc.Htlc static method), 139  
 objectid\_valid() (bitshares.proposal.Proposal static method), 160  
 objectid\_valid() (bitshares.vesting.Vesting static method), 168  
 objectid\_valid() (bitshares.witness.Witness static method), 172  
 objectid\_valid() (bitshares.worker.Worker static method), 176  
 ObjectNotInProposalBuffer, 135  
 op (bitsharesbase.objects.Operation attribute), 188  
 Op\_wrapper (class in bitsharesbase.operations), 215  
 openorders (bitshares.account.Account attribute), 99  
 openorders (bitshares.aio.account.Account attribute), 21  
 operation (bitsharesbase.objects.Operation attribute), 188  
 Operation (class in bitsharesbase.objects), 188  
 operation\_klass (bitsharesbase.signedtransactions.Signed\_Transaction attribute), 224  
 operations (bitsharesbase.objects.Operation attribute), 188  
 opId (bitsharesbase.objects.Operation attribute), 188  
 ops (bitsharesbase.objects.Operation attribute), 188

ops (in module bitsharesbase.operationids), 191  
 ops () (bitshares.aio.blockchain.Blockchain method), 47  
 ops () (bitshares.blockchain.Blockchain method), 125  
 Order (class in bitshares.aio.price), 71  
 Order (class in bitshares.price), 152  
 orderbook () (bitshares.aio.market.Market method), 64  
 orderbook () (bitshares.market.Market method), 144  
 Override\_transfer (class in bitsharesbase.operations), 216

## P

participation\_rate (bitshares.aio.blockchain.Blockchain attribute), 47  
 participation\_rate (bitshares.blockchain.Blockchain attribute), 126  
 PasswordKey (class in bitsharesbase.account), 180  
 perform\_id\_tests (bitshares.account.Account attribute), 99  
 perform\_id\_tests (bitshares.aio.account.Account attribute), 21  
 perform\_id\_tests (bitshares.aio.asset.Asset attribute), 28  
 perform\_id\_tests (bitshares.aio.block.Block attribute), 42  
 perform\_id\_tests (bitshares.aio.block.BlockHeader attribute), 44  
 perform\_id\_tests (bitshares.aio.blockchainobject.BlockchainObject attribute), 48  
 perform\_id\_tests (bitshares.aio.blockchainobject.Object attribute), 50  
 perform\_id\_tests (bitshares.aio.committee.Committee attribute), 52  
 perform\_id\_tests (bitshares.aio.genesisbalance.GenesisBalance attribute), 57  
 perform\_id\_tests (bitshares.aio.htlc.Htlc attribute), 60  
 perform\_id\_tests (bitshares.aio.proposal.Proposal attribute), 79  
 perform\_id\_tests (bitshares.aio.vesting.Vesting attribute), 87  
 perform\_id\_tests (bitshares.aio.witness.Witness attribute), 91  
 perform\_id\_tests (bitshares.aio.worker.Worker attribute), 94  
 perform\_id\_tests (bitshares.asset.Asset attribute), 106

- `perform_id_tests` (*bitshares.block.Block* attribute), 121
- `perform_id_tests` (*bitshares.block.BlockHeader* attribute), 123
- `perform_id_tests` (*bitshares.blockchainobject.BlockchainObject* attribute), 127
- `perform_id_tests` (*bitshares.blockchainobject.Object* attribute), 129
- `perform_id_tests` (*bitshares.committee.Committee* attribute), 131
- `perform_id_tests` (*bitshares.genesisbalance.GenesisBalance* attribute), 136
- `perform_id_tests` (*bitshares.htlc.Htlc* attribute), 139
- `perform_id_tests` (*bitshares.proposal.Proposal* attribute), 161
- `perform_id_tests` (*bitshares.vesting.Vesting* attribute), 168
- `perform_id_tests` (*bitshares.witness.Witness* attribute), 172
- `perform_id_tests` (*bitshares.worker.Worker* attribute), 176
- `Permission` (class in *bitsharesbase.objects*), 189
- `permission_types` (*bitshares.aio.transactionbuilder.TransactionBuilder* attribute), 85
- `permission_types` (*bitshares.transactionbuilder.TransactionBuilder* attribute), 166
- `permissions` (*bitshares.aio.asset.Asset* attribute), 28
- `permissions` (*bitshares.asset.Asset* attribute), 106
- `point()` (*bitsharesbase.account.PublicKey* method), 182
- `pop()` (*bitshares.account.Account* method), 99
- `pop()` (*bitshares.account.AccountUpdate* method), 101
- `pop()` (*bitshares.aio.account.Account* method), 21
- `pop()` (*bitshares.aio.account.AccountUpdate* method), 23
- `pop()` (*bitshares.aio.amount.Amount* method), 25
- `pop()` (*bitshares.aio.asset.Asset* method), 28
- `pop()` (*bitshares.aio.block.Block* method), 42
- `pop()` (*bitshares.aio.block.BlockHeader* method), 44
- `pop()` (*bitshares.aio.blockchainobject.BlockchainObject* method), 48
- `pop()` (*bitshares.aio.blockchainobject.Object* method), 50
- `pop()` (*bitshares.aio.committee.Committee* method), 52
- `pop()` (*bitshares.aio.genesisbalance.GenesisBalance* method), 57
- `pop()` (*bitshares.aio.genesisbalance.GenesisBalances* method), 58
- `pop()` (*bitshares.aio.htlc.Htlc* method), 60
- `pop()` (*bitshares.aio.market.Market* method), 65
- `pop()` (*bitshares.aio.price.FilledOrder* method), 70
- `pop()` (*bitshares.aio.price.Order* method), 72
- `pop()` (*bitshares.aio.price.Price* method), 75
- `pop()` (*bitshares.aio.price.PriceFeed* method), 76
- `pop()` (*bitshares.aio.price.UpdateCallOrder* method), 77
- `pop()` (*bitshares.aio.proposal.Proposal* method), 79
- `pop()` (*bitshares.aio.proposal.Proposals* method), 81
- `pop()` (*bitshares.aio.transactionbuilder.TransactionBuilder* method), 85
- `pop()` (*bitshares.aio.vesting.Vesting* method), 87
- `pop()` (*bitshares.aio.witness.Witness* method), 91
- `pop()` (*bitshares.aio.witness.Witnesses* method), 93
- `pop()` (*bitshares.aio.worker.Worker* method), 94
- `pop()` (*bitshares.aio.worker.Workers* method), 96
- `pop()` (*bitshares.amount.Amount* method), 103
- `pop()` (*bitshares.asset.Asset* method), 106
- `pop()` (*bitshares.block.Block* method), 121
- `pop()` (*bitshares.block.BlockHeader* method), 123
- `pop()` (*bitshares.blockchainobject.BlockchainObject* method), 127
- `pop()` (*bitshares.blockchainobject.Object* method), 129
- `pop()` (*bitshares.committee.Committee* method), 131
- `pop()` (*bitshares.genesisbalance.GenesisBalance* method), 136
- `pop()` (*bitshares.genesisbalance.GenesisBalances* method), 138
- `pop()` (*bitshares.htlc.Htlc* method), 139
- `pop()` (*bitshares.market.Market* method), 145
- `pop()` (*bitshares.price.FilledOrder* method), 152
- `pop()` (*bitshares.price.Order* method), 153
- `pop()` (*bitshares.price.Price* method), 156
- `pop()` (*bitshares.price.PriceFeed* method), 157
- `pop()` (*bitshares.price.UpdateCallOrder* method), 159
- `pop()` (*bitshares.proposal.Proposal* method), 161
- `pop()` (*bitshares.proposal.Proposals* method), 163
- `pop()` (*bitshares.transactionbuilder.TransactionBuilder* method), 166
- `pop()` (*bitshares.vesting.Vesting* method), 168
- `pop()` (*bitshares.witness.Witness* method), 172
- `pop()` (*bitshares.witness.Witnesses* method), 174
- `pop()` (*bitshares.worker.Worker* method), 176
- `pop()` (*bitshares.worker.Workers* method), 178
- `pop()` (*bitsharesbase.objects.AccountCreateExtensions.Buyback\_options* method), 183
- `pop()` (*bitsharesbase.objects.AccountCreateExtensions.Null\_ext* method), 184
- `pop()` (*bitsharesbase.objects.AccountOptions* method), 185
- `pop()` (*bitsharesbase.objects.AssetOptions* method), 186



- pop () (*bitsharesbase.objects.BitAssetOptions* method), 186
- pop () (*bitsharesbase.objects.Memo* method), 187
- pop () (*bitsharesbase.objects.Operation* method), 188
- pop () (*bitsharesbase.objects.Permission* method), 189
- pop () (*bitsharesbase.objects.Price* method), 190
- pop () (*bitsharesbase.objects.PriceFeed* method), 191
- pop () (*bitsharesbase.operations.Account\_create* method), 192
- pop () (*bitsharesbase.operations.Account\_update* method), 192
- pop () (*bitsharesbase.operations.Account\_upgrade* method), 193
- pop () (*bitsharesbase.operations.Account\_whitelist* method), 194
- pop () (*bitsharesbase.operations.Assert* method), 195
- pop () (*bitsharesbase.operations.Asset\_claim\_fees* method), 195
- pop () (*bitsharesbase.operations.Asset\_claim\_pool* method), 196
- pop () (*bitsharesbase.operations.Asset\_create* method), 197
- pop () (*bitsharesbase.operations.Asset\_fund\_fee\_pool* method), 198
- pop () (*bitsharesbase.operations.Asset\_global\_settle* method), 198
- pop () (*bitsharesbase.operations.Asset\_issue* method), 199
- pop () (*bitsharesbase.operations.Asset\_publish\_feed* method), 200
- pop () (*bitsharesbase.operations.Asset\_reserve* method), 200
- pop () (*bitsharesbase.operations.Asset\_settle* method), 201
- pop () (*bitsharesbase.operations.Asset\_update* method), 202
- pop () (*bitsharesbase.operations.Asset\_update\_bitasset* method), 203
- pop () (*bitsharesbase.operations.Asset\_update\_feed\_producers* method), 203
- pop () (*bitsharesbase.operations.Asset\_update\_issuer* method), 204
- pop () (*bitsharesbase.operations.Balance\_claim* method), 205
- pop () (*bitsharesbase.operations.Bid\_collateral* method), 205
- pop () (*bitsharesbase.operations.Call\_order\_update* method), 206
- pop () (*bitsharesbase.operations.Committee\_member\_create* method), 207
- pop () (*bitsharesbase.operations.Custom* method), 208
- pop () (*bitsharesbase.operations.Htlc\_create* method), 208
- pop () (*bitsharesbase.operations.Htlc\_extend* method), 209
- pop () (*bitsharesbase.operations.Htlc\_redeem* method), 210
- pop () (*bitsharesbase.operations.Limit\_order\_cancel* method), 211
- pop () (*bitsharesbase.operations.Limit\_order\_create* method), 211
- pop () (*bitsharesbase.operations.Liquidity\_pool\_create* method), 212
- pop () (*bitsharesbase.operations.Liquidity\_pool\_delete* method), 213
- pop () (*bitsharesbase.operations.Liquidity\_pool\_deposit* method), 213
- pop () (*bitsharesbase.operations.Liquidity\_pool\_exchange* method), 214
- pop () (*bitsharesbase.operations.Liquidity\_pool\_withdraw* method), 215
- pop () (*bitsharesbase.operations.Op\_wrapper* method), 216
- pop () (*bitsharesbase.operations.Override\_transfer* method), 216
- pop () (*bitsharesbase.operations.Proposal\_create* method), 217
- pop () (*bitsharesbase.operations.Proposal\_update* method), 218
- pop () (*bitsharesbase.operations.Ticket\_create\_operation* method), 218
- pop () (*bitsharesbase.operations.Ticket\_update\_operation* method), 219
- pop () (*bitsharesbase.operations.Transfer* method), 220
- pop () (*bitsharesbase.operations.Vesting\_balance\_withdraw* method), 221
- pop () (*bitsharesbase.operations.Withdraw\_permission\_create* method), 221
- pop () (*bitsharesbase.operations.Witness\_update* method), 222
- pop () (*bitsharesbase.operations.Worker\_create* method), 223
- popitem () (*bitshares.signedtransactions.Signed\_Transaction* method), 224
- popitem () (*bitshares.account.Account* method), 100
- popitem () (*bitshares.account.AccountUpdate* method), 101
- popitem () (*bitshares.aio.account.Account* method), 21
- popitem () (*bitshares.aio.account.AccountUpdate* method), 23
- popitem () (*bitshares.aio.amount.Amount* method), 25
- popitem () (*bitshares.aio.asset.Asset* method), 28
- popitem () (*bitshares.aio.block.Block* method), 42
- popitem () (*bitshares.aio.block.BlockHeader* method), 44
- popitem () (*bitshares.aio.blockchainobject.BlockchainObject* method), 49
- popitem () (*bitshares.aio.blockchainobject.Object* method), 49

method), 50  
popitem() (bitshares.aio.committee.Committee method), 52  
popitem() (bitshares.aio.genesisbalance.GenesisBalance method), 57  
popitem() (bitshares.aio.htlc.Htlc method), 60  
popitem() (bitshares.aio.market.Market method), 65  
popitem() (bitshares.aio.price.FilledOrder method), 71  
popitem() (bitshares.aio.price.Order method), 72  
popitem() (bitshares.aio.price.Price method), 75  
popitem() (bitshares.aio.price.PriceFeed method), 76  
popitem() (bitshares.aio.price.UpdateCallOrder method), 78  
popitem() (bitshares.aio.proposal.Proposal method), 79  
popitem() (bitshares.aio.transactionbuilder.TransactionBuilder method), 85  
popitem() (bitshares.aio.vesting.Vesting method), 87  
popitem() (bitshares.aio.witness.Witness method), 91  
popitem() (bitshares.aio.worker.Worker method), 94  
popitem() (bitshares.amount.Amount method), 103  
popitem() (bitshares.asset.Asset method), 106  
popitem() (bitshares.block.Block method), 121  
popitem() (bitshares.block.BlockHeader method), 123  
popitem() (bitshares.blockchainobject.BlockchainObject method), 128  
popitem() (bitshares.blockchainobject.Object method), 129  
popitem() (bitshares.committee.Committee method), 131  
popitem() (bitshares.genesisbalance.GenesisBalance method), 136  
popitem() (bitshares.htlc.Htlc method), 139  
popitem() (bitshares.market.Market method), 145  
popitem() (bitshares.price.FilledOrder method), 152  
popitem() (bitshares.price.Order method), 154  
popitem() (bitshares.price.Price method), 156  
popitem() (bitshares.price.PriceFeed method), 157  
popitem() (bitshares.price.UpdateCallOrder method), 159  
popitem() (bitshares.proposal.Proposal method), 161  
popitem() (bitshares.transactionbuilder.TransactionBuilder method), 166  
popitem() (bitshares.vesting.Vesting method), 168  
popitem() (bitshares.witness.Witness method), 172  
popitem() (bitshares.worker.Worker method), 176  
popitem() (bitsharesbase.objects.AccountCreateExtensions.Buyback\_options method), 183  
popitem() (bitsharesbase.objects.AccountCreateExtensions.Null\_ext method), 184  
popitem() (bitsharesbase.objects.AccountOptions method), 185  
popitem() (bitsharesbase.objects.AssetOptions method), 186  
popitem() (bitsharesbase.objects.BitAssetOptions method), 186  
popitem() (bitsharesbase.objects.Memo method), 187  
popitem() (bitsharesbase.objects.Permission method), 189  
popitem() (bitsharesbase.objects.Price method), 190  
popitem() (bitsharesbase.objects.PriceFeed method), 191  
popitem() (bitsharesbase.operations.Account\_create method), 192  
popitem() (bitsharesbase.operations.Account\_update method), 192  
popitem() (bitsharesbase.operations.Account\_upgrade method), 193  
popitem() (bitsharesbase.operations.Account\_whitelist method), 194  
popitem() (bitsharesbase.operations.Assert method), 195  
popitem() (bitsharesbase.operations.Asset\_claim\_fees method), 195  
popitem() (bitsharesbase.operations.Asset\_claim\_pool method), 196  
popitem() (bitsharesbase.operations.Asset\_create method), 197  
popitem() (bitsharesbase.operations.Asset\_fund\_fee\_pool method), 198  
popitem() (bitsharesbase.operations.Asset\_global\_settle method), 198  
popitem() (bitsharesbase.operations.Asset\_issue method), 199  
popitem() (bitsharesbase.operations.Asset\_publish\_feed method), 200  
popitem() (bitsharesbase.operations.Asset\_reserve method), 200  
popitem() (bitsharesbase.operations.Asset\_settle method), 201  
popitem() (bitsharesbase.operations.Asset\_update method), 202  
popitem() (bitsharesbase.operations.Asset\_update\_bitasset method), 203  
popitem() (bitsharesbase.operations.Asset\_update\_feed\_producers method), 203

`popitem()` (*bitsharesbase.operations.Asset\_update\_issuer method*), 204  
`popitem()` (*bitsharesbase.operations.Balance\_claim method*), 205  
`popitem()` (*bitsharesbase.operations.Bid\_collateral method*), 206  
`popitem()` (*bitsharesbase.operations.Call\_order\_update method*), 206  
`popitem()` (*bitsharesbase.operations.Committee\_member\_create method*), 207  
`popitem()` (*bitsharesbase.operations.Custom method*), 208  
`popitem()` (*bitsharesbase.operations.Htlc\_create method*), 208  
`popitem()` (*bitsharesbase.operations.Htlc\_extend method*), 209  
`popitem()` (*bitsharesbase.operations.Htlc\_redeem method*), 210  
`popitem()` (*bitsharesbase.operations.Limit\_order\_cancel method*), 211  
`popitem()` (*bitsharesbase.operations.Limit\_order\_create method*), 211  
`popitem()` (*bitsharesbase.operations.Liquidity\_pool\_create method*), 212  
`popitem()` (*bitsharesbase.operations.Liquidity\_pool\_delete method*), 213  
`popitem()` (*bitsharesbase.operations.Liquidity\_pool\_deposit method*), 214  
`popitem()` (*bitsharesbase.operations.Liquidity\_pool\_exchange method*), 214  
`popitem()` (*bitsharesbase.operations.Liquidity\_pool\_withdraw method*), 215  
`popitem()` (*bitsharesbase.operations.Op\_wrapper method*), 216  
`popitem()` (*bitsharesbase.operations.Override\_transfer method*), 216  
`popitem()` (*bitsharesbase.operations.Proposal\_create method*), 217  
`popitem()` (*bitsharesbase.operations.Proposal\_update method*), 218  
`popitem()` (*bitsharesbase.operations.Ticket\_create\_operation method*), 219  
`popitem()` (*bitsharesbase.operations.Ticket\_update\_operation method*), 219  
`popitem()` (*bitsharesbase.operations.Transfer method*), 220  
`popitem()` (*bitsharesbase.operations.Vesting\_balance\_withdraw method*), 221  
`popitem()` (*bitsharesbase.operations.Withdraw\_permission\_create method*), 221  
`popitem()` (*bitsharesbase.operations.Witness\_update method*), 222  
`popitem()` (*bitsharesbase.operations.Worker\_create method*), 223  
`popitem()` (*bitsharesbase.signedtransactions.Signed\_Transaction method*), 224  
`post_format()` (*bitshares.aio.worker.Worker method*), 95  
`post_format()` (*bitshares.worker.Worker method*), 176  
`precision` (*bitshares.aio.asset.Asset attribute*), 28  
`precision` (*bitshares.asset.Asset attribute*), 106  
`prefix` (*bitshares.aio.bitshares.BitShares attribute*), 37  
`prefix` (*bitshares.aio.wallet.Wallet attribute*), 89  
`prefix` (*bitshares.bitshares.BitShares attribute*), 117  
`prefix` (*bitshares.wallet.Wallet attribute*), 170  
`prefix` (*bitsharesbase.account.Address attribute*), 179  
`prefix` (*bitsharesbase.account.BrainKey attribute*), 180  
`prefix` (*bitsharesbase.account.PasswordKey attribute*), 180  
`prefix` (*bitsharesbase.account.PrivateKey attribute*), 181  
`prefix` (*bitsharesbase.account.PublicKey attribute*), 182  
`price` (*bitshares.price.Order attribute*), 154  
`Price` (*class in bitshares.aio.price*), 73  
`Price` (*class in bitshares.price*), 154  
`Price` (*class in bitsharesbase.objects*), 189  
`PriceFeed` (*class in bitshares.aio.price*), 75  
`PriceFeed` (*class in bitshares.price*), 157  
`PriceFeed` (*class in bitsharesbase.objects*), 190  
`PrivateKey` (*class in bitsharesbase.account*), 180  
`privatekey()` (*bitshares.aio.wallet.Wallet method*), 89  
`privatekey()` (*bitshares.wallet.Wallet method*), 171  
`process_account()` (*bitshares.notify.Notify method*), 150  
`process_market()` (*bitshares.notify.Notify method*), 150  
`propbuffer` (*bitshares.aio.bitshares.BitShares attribute*), 37  
`propbuffer` (*bitshares.bitshares.BitShares attribute*),

117  
 Proposal (class in *bitshares.aio.proposal*), 78  
 Proposal (class in *bitshares.proposal*), 160  
 proposal() (*bitshares.aio.bitshares.BitShares* method), 37  
 proposal() (*bitshares.bitshares.BitShares* method), 117  
 Proposal\_create (class in *bitshares-base.operations*), 217  
 Proposal\_update (class in *bitshares-base.operations*), 217  
 ProposalBuilder (class in *bitshares.aio.transactionbuilder*), 82  
 ProposalBuilder (class in *bitshares.transactionbuilder*), 164  
 Proposals (class in *bitshares.aio.proposal*), 80  
 Proposals (class in *bitshares.proposal*), 162  
 proposed\_operations (*bitshares.aio.proposal.Proposal* attribute), 79  
 proposed\_operations (*bitshares.proposal.Proposal* attribute), 161  
 proposer (*bitshares.aio.proposal.Proposal* attribute), 79  
 proposer (*bitshares.proposal.Proposal* attribute), 161  
 pubkey (*bitsharesbase.account.PrivateKey* attribute), 181  
 pubkey (*bitsharesbase.account.PublicKey* attribute), 182  
 PublicKey (class in *bitsharesbase.account*), 181  
 publickey\_from\_wif() (*bitshares.aio.wallet.Wallet* method), 89  
 publickey\_from\_wif() (*bitshares.wallet.Wallet* method), 171  
 publish\_price\_feed() (*bitshares.aio.bitshares.BitShares* method), 37  
 publish\_price\_feed() (*bitshares.bitshares.BitShares* method), 117

## R

refresh() (*bitshares.account.Account* method), 100  
 refresh() (*bitshares.aio.account.Account* method), 21  
 refresh() (*bitshares.aio.asset.Asset* method), 28  
 refresh() (*bitshares.aio.block.Block* method), 42  
 refresh() (*bitshares.aio.block.BlockHeader* method), 44  
 refresh() (*bitshares.aio.blockchainobject.Object* method), 50  
 refresh() (*bitshares.aio.committee.Committee* method), 52  
 refresh() (*bitshares.aio.genesisbalance.GenesisBalance* method), 57  
 refresh() (*bitshares.aio.htlc.Htlc* method), 60  
 refresh() (*bitshares.aio.proposal.Proposal* method), 79  
 refresh() (*bitshares.aio.proposal.Proposals* method), 81  
 refresh() (*bitshares.aio.vesting.Vesting* method), 87  
 refresh() (*bitshares.aio.witness.Witness* method), 91  
 refresh() (*bitshares.aio.witness.Witnesses* method), 93  
 refresh() (*bitshares.aio.worker.Worker* method), 95  
 refresh() (*bitshares.aio.worker.Workers* method), 97  
 refresh() (*bitshares.asset.Asset* method), 106  
 refresh() (*bitshares.block.Block* method), 121  
 refresh() (*bitshares.block.BlockHeader* method), 123  
 refresh() (*bitshares.blockchainobject.Object* method), 129  
 refresh() (*bitshares.committee.Committee* method), 131  
 refresh() (*bitshares.genesisbalance.GenesisBalance* method), 136  
 refresh() (*bitshares.htlc.Htlc* method), 139  
 refresh() (*bitshares.proposal.Proposal* method), 161  
 refresh() (*bitshares.proposal.Proposals* method), 163  
 refresh() (*bitshares.vesting.Vesting* method), 168  
 refresh() (*bitshares.witness.Witness* method), 172  
 refresh() (*bitshares.witness.Witnesses* method), 174  
 refresh() (*bitshares.worker.Worker* method), 176  
 refresh() (*bitshares.worker.Workers* method), 178  
 registrar (*bitshares.aio.bitshares.BitShares* attribute), 38  
 release() (*bitshares.aio.asset.Asset* method), 28  
 release() (*bitshares.asset.Asset* method), 106  
 remove() (*bitshares.aio.genesisbalance.GenesisBalances* method), 58  
 remove() (*bitshares.aio.proposal.Proposals* method), 81  
 remove() (*bitshares.aio.witness.Witnesses* method), 93  
 remove() (*bitshares.aio.worker.Workers* method), 97  
 remove() (*bitshares.genesisbalance.GenesisBalances* method), 138  
 remove() (*bitshares.proposal.Proposals* method), 163  
 remove() (*bitshares.witness.Witnesses* method), 174  
 remove() (*bitshares.worker.Workers* method), 178  
 remove() (*bitsharesbase.objects.Operation* method), 188  
 remove\_authorities() (*bitshares.aio.asset.Asset* method), 28  
 remove\_authorities() (*bitshares.asset.Asset* method), 107  
 remove\_markets() (*bitshares.aio.asset.Asset* method), 28  
 remove\_markets() (*bitshares.asset.Asset* method), 107  
 removeAccount() (*bitshares.aio.wallet.Wallet*

method), 89  
 removeAccount() (bitshares.wallet.Wallet method), 171  
 removePrivateKeyFromPublicKey() (bitshares.aio.wallet.Wallet method), 89  
 removePrivateKeyFromPublicKey() (bitshares.wallet.Wallet method), 171  
 reserve() (bitshares.aio.bitshares.BitShares method), 38  
 reserve() (bitshares.bitshares.BitShares method), 117  
 reset\_subscriptions() (bitshares.notify.Notify method), 150  
 returnFees() (bitshares.aio.dex.Dex method), 55  
 returnFees() (bitshares.dex.Dex method), 134  
 reverse() (bitshares.aio.genesisbalance.GenesisBalances method), 58  
 reverse() (bitshares.aio.proposal.Proposals method), 81  
 reverse() (bitshares.aio.witness.Witnesses method), 93  
 reverse() (bitshares.aio.worker.Workers method), 97  
 reverse() (bitshares.genesisbalance.GenesisBalances method), 138  
 reverse() (bitshares.proposal.Proposals method), 163  
 reverse() (bitshares.witness.Witnesses method), 174  
 reverse() (bitshares.worker.Workers method), 178  
 reverse() (bitsharesbase.objects.Operation method), 189  
 review\_period (bitshares.aio.proposal.Proposal attribute), 79  
 review\_period (bitshares.proposal.Proposal attribute), 161  
 rpc (bitshares.aio.wallet.Wallet attribute), 89  
 rpc (bitshares.wallet.Wallet attribute), 171  
 RPCConnectionRequired, 135

## S

seize() (bitshares.aio.asset.Asset method), 28  
 seize() (bitshares.asset.Asset method), 107  
 sell() (bitshares.aio.market.Market method), 65  
 sell() (bitshares.market.Market method), 145  
 set() (bitsharesbase.objects.Operation method), 189  
 set\_blocking() (bitshares.aio.bitshares.BitShares method), 38  
 set\_blocking() (bitshares.bitshares.BitShares method), 117  
 set\_cache\_store() (bitshares.account.Account static method), 100  
 set\_cache\_store() (bitshares.aio.account.Account static method), 21  
 set\_cache\_store() (bitshares.aio.asset.Asset static method), 29

set\_cache\_store() (bitshares.aio.block.Block static method), 42  
 set\_cache\_store() (bitshares.aio.block.BlockHeader static method), 44  
 set\_cache\_store() (bitshares.aio.blockchainobject.BlockchainObject static method), 49  
 set\_cache\_store() (bitshares.aio.blockchainobject.Object static method), 50  
 set\_cache\_store() (bitshares.aio.committee.Committee static method), 52  
 set\_cache\_store() (bitshares.aio.genesisbalance.GenesisBalance static method), 57  
 set\_cache\_store() (bitshares.aio.htlc.Htlc static method), 60  
 set\_cache\_store() (bitshares.aio.proposal.Proposal static method), 79  
 set\_cache\_store() (bitshares.aio.proposal.Proposals static method), 82  
 set\_cache\_store() (bitshares.aio.vesting.Vesting static method), 87  
 set\_cache\_store() (bitshares.aio.witness.Witness static method), 91  
 set\_cache\_store() (bitshares.aio.witness.Witnesses static method), 93  
 set\_cache\_store() (bitshares.aio.worker.Worker static method), 95  
 set\_cache\_store() (bitshares.aio.worker.Workers static method), 97  
 set\_cache\_store() (bitshares.asset.Asset static method), 107  
 set\_cache\_store() (bitshares.block.Block static method), 121  
 set\_cache\_store() (bitshares.block.BlockHeader static method), 123  
 set\_cache\_store() (bitshares.blockchainobject.BlockchainObject static method), 128  
 set\_cache\_store() (bitshares.blockchainobject.Object static method), 129  
 set\_cache\_store() (bitshares.committee.Committee static method), 131  
 set\_cache\_store() (bitshares.genesisbalance.GenesisBalance static method), 136



`set_cache_store()` (*bitshares.htlc.Htlc static method*), 140

`set_cache_store()` (*bitshares.proposal.Proposal static method*), 161

`set_cache_store()` (*bitshares.proposal.Proposals static method*), 163

`set_cache_store()` (*bitshares.vesting.Vesting static method*), 168

`set_cache_store()` (*bitshares.witness.Witness static method*), 172

`set_cache_store()` (*bitshares.witness.Witnesses static method*), 175

`set_cache_store()` (*bitshares.worker.Worker static method*), 176

`set_cache_store()` (*bitshares.worker.Workers static method*), 178

`set_default_account()` (*bitshares.aio.bitshares.BitShares method*), 38

`set_default_account()` (*bitshares.bitshares.BitShares method*), 117

`set_expiration()` (*bitshares.aio.transactionbuilder.ProposalBuilder method*), 83

`set_expiration()` (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 85

`set_expiration()` (*bitshares.transactionbuilder.ProposalBuilder method*), 164

`set_expiration()` (*bitshares.transactionbuilder.TransactionBuilder method*), 166

`set_fee_asset()` (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 85

`set_fee_asset()` (*bitshares.transactionbuilder.TransactionBuilder method*), 166

`set_market_fee()` (*bitshares.aio.asset.Asset method*), 29

`set_market_fee()` (*bitshares.asset.Asset method*), 107

`set_parent()` (*bitshares.aio.transactionbuilder.ProposalBuilder method*), 83

`set_parent()` (*bitshares.transactionbuilder.ProposalBuilder method*), 165

`set_prefix()` (*bitsharesbase.account.Address method*), 179

`set_prefix()` (*bitsharesbase.account.BrainKey method*), 180

`set_prefix()` (*bitsharesbase.account.PasswordKey method*), 180

`set_prefix()` (*bitsharesbase.account.PrivateKey method*), 181

`set_prefix()` (*bitsharesbase.account.PublicKey method*), 182

`set_proposer()` (*bitshares.aio.transactionbuilder.ProposalBuilder method*), 83

`set_proposer()` (*bitshares.transactionbuilder.ProposalBuilder method*), 165

`set_proxy()` (*bitshares.aio.bitshares.BitShares method*), 38

`set_proxy()` (*bitshares.bitshares.BitShares method*), 117

`set_review()` (*bitshares.aio.transactionbuilder.ProposalBuilder method*), 83

`set_review()` (*bitshares.transactionbuilder.ProposalBuilder method*), 165

`set_shared_bitshares_instance()` (*in module bitshares.aio.instance*), 61

`set_shared_bitshares_instance()` (*in module bitshares.instance*), 141

`set_shared_blockchain_instance()` (*bitshares.account.Account class method*), 100

`set_shared_blockchain_instance()` (*bitshares.account.AccountUpdate class method*), 101

`set_shared_blockchain_instance()` (*bitshares.aio.account.Account class method*), 22

`set_shared_blockchain_instance()` (*bitshares.aio.account.AccountUpdate class method*), 23

`set_shared_blockchain_instance()` (*bitshares.aio.amount.Amount class method*), 25

`set_shared_blockchain_instance()` (*bitshares.aio.asset.Asset class method*), 29

`set_shared_blockchain_instance()` (*bitshares.aio.block.Block class method*), 42

`set_shared_blockchain_instance()` (*bitshares.aio.block.BlockHeader class method*), 44

`set_shared_blockchain_instance()` (*bitshares.aio.blockchain.Blockchain class method*), 47

`set_shared_blockchain_instance()` (*bitshares.aio.blockchainobject.BlockchainObject class method*), 49

`set_shared_blockchain_instance()` (*bitshares.aio.blockchainobject.Object class method*), 50

`set_shared_blockchain_instance()` (*bitshares.aio.committee.Committee class method*), 52

`set_shared_blockchain_instance()` (*bitshares.aio.dex.Dex class method*), 55  
`set_shared_blockchain_instance()` (*bitshares.aio.genesisbalance.GenesisBalance class method*), 57  
`set_shared_blockchain_instance()` (*bitshares.aio.genesisbalance.GenesisBalances class method*), 58  
`set_shared_blockchain_instance()` (*bitshares.aio.htlc.Htlc class method*), 60  
`set_shared_blockchain_instance()` (*bitshares.aio.instance.BlockchainInstance class method*), 61  
`set_shared_blockchain_instance()` (*bitshares.aio.market.Market class method*), 65  
`set_shared_blockchain_instance()` (*bitshares.aio.memo.Memo class method*), 68  
`set_shared_blockchain_instance()` (*bitshares.aio.message.Message class method*), 69  
`set_shared_blockchain_instance()` (*bitshares.aio.price.FilledOrder class method*), 71  
`set_shared_blockchain_instance()` (*bitshares.aio.price.Order class method*), 72  
`set_shared_blockchain_instance()` (*bitshares.aio.price.Price class method*), 75  
`set_shared_blockchain_instance()` (*bitshares.aio.price.PriceFeed class method*), 76  
`set_shared_blockchain_instance()` (*bitshares.aio.price.UpdateCallOrder class method*), 78  
`set_shared_blockchain_instance()` (*bitshares.aio.proposal.Proposal class method*), 79  
`set_shared_blockchain_instance()` (*bitshares.aio.proposal.Proposals class method*), 82  
`set_shared_blockchain_instance()` (*bitshares.aio.transactionbuilder.ProposalBuilder class method*), 83  
`set_shared_blockchain_instance()` (*bitshares.aio.transactionbuilder.TransactionBuilder class method*), 85  
`set_shared_blockchain_instance()` (*bitshares.aio.vesting.Vesting class method*), 87  
`set_shared_blockchain_instance()` (*bitshares.aio.wallet.Wallet class method*), 89  
`set_shared_blockchain_instance()` (*bitshares.aio.witness.Witness class method*), 91  
`set_shared_blockchain_instance()` (*bitshares.aio.witness.Witnesses class method*), 93  
`set_shared_blockchain_instance()` (*bitshares.aio.worker.Worker class method*), 95  
`set_shared_blockchain_instance()` (*bitshares.aio.worker.Workers class method*), 97  
`set_shared_blockchain_instance()` (*bitshares.amount.Amount class method*), 103  
`set_shared_blockchain_instance()` (*bitshares.asset.Asset class method*), 107  
`set_shared_blockchain_instance()` (*bitshares.block.Block class method*), 121  
`set_shared_blockchain_instance()` (*bitshares.block.BlockHeader class method*), 123  
`set_shared_blockchain_instance()` (*bitshares.blockchain.Blockchain class method*), 126  
`set_shared_blockchain_instance()` (*bitshares.blockchainobject.BlockchainObject class method*), 128  
`set_shared_blockchain_instance()` (*bitshares.blockchainobject.Object class method*), 129  
`set_shared_blockchain_instance()` (*bitshares.committee.Committee class method*), 131  
`set_shared_blockchain_instance()` (*bitshares.dex.Dex class method*), 134  
`set_shared_blockchain_instance()` (*bitshares.genesisbalance.GenesisBalance class method*), 136  
`set_shared_blockchain_instance()` (*bitshares.genesisbalance.GenesisBalances class method*), 138  
`set_shared_blockchain_instance()` (*bitshares.htlc.Htlc class method*), 140  
`set_shared_blockchain_instance()` (*bitshares.instance.BlockchainInstance class method*), 141  
`set_shared_blockchain_instance()` (*bitshares.market.Market class method*), 145  
`set_shared_blockchain_instance()` (*bitshares.memo.Memo class method*), 148  
`set_shared_blockchain_instance()` (*bitshares.message.Message class method*), 148  
`set_shared_blockchain_instance()` (*bitshares.notify.Notify class method*), 150  
`set_shared_blockchain_instance()` (*bitshares.price.FilledOrder class method*), 152  
`set_shared_blockchain_instance()` (*bit-*

- shares.price.Order class method*), 154
- `set_shared_blockchain_instance()` (*bit-shares.price.Price class method*), 156
- `set_shared_blockchain_instance()` (*bit-shares.price.PriceFeed class method*), 157
- `set_shared_blockchain_instance()` (*bit-shares.price.UpdateCallOrder class method*), 159
- `set_shared_blockchain_instance()` (*bit-shares.proposal.Proposal class method*), 161
- `set_shared_blockchain_instance()` (*bit-shares.proposal.Proposals class method*), 163
- `set_shared_blockchain_instance()` (*bit-shares.transactionbuilder.ProposalBuilder class method*), 165
- `set_shared_blockchain_instance()` (*bit-shares.transactionbuilder.TransactionBuilder class method*), 166
- `set_shared_blockchain_instance()` (*bit-shares.vesting.Vesting class method*), 168
- `set_shared_blockchain_instance()` (*bit-shares.wallet.Wallet class method*), 171
- `set_shared_blockchain_instance()` (*bit-shares.witness.Witness class method*), 172
- `set_shared_blockchain_instance()` (*bit-shares.witness.Witnesses class method*), 175
- `set_shared_blockchain_instance()` (*bit-shares.worker.Worker class method*), 176
- `set_shared_blockchain_instance()` (*bit-shares.worker.Workers class method*), 178
- `set_shared_blockchain_instance()` (*in module bitshares.aio.instance*), 61
- `set_shared_blockchain_instance()` (*in module bitshares.instance*), 141
- `set_shared_config()` (*bitshares.account.Account class method*), 100
- `set_shared_config()` (*bit-shares.account.AccountUpdate class method*), 101
- `set_shared_config()` (*bit-shares.aio.account.Account class method*), 22
- `set_shared_config()` (*bit-shares.aio.account.AccountUpdate class method*), 23
- `set_shared_config()` (*bit-shares.aio.amount.Amount class method*), 25
- `set_shared_config()` (*bitshares.aio.asset.Asset class method*), 29
- `set_shared_config()` (*bitshares.aio.block.Block class method*), 42
- `set_shared_config()` (*bit-shares.aio.block.BlockHeader class method*), 44
- `set_shared_config()` (*bit-shares.aio.blockchain.Blockchain class method*), 47
- `set_shared_config()` (*bit-shares.aio.blockchainobject.BlockchainObject class method*), 49
- `set_shared_config()` (*bit-shares.aio.blockchainobject.Object class method*), 50
- `set_shared_config()` (*bit-shares.aio.committee.Committee class method*), 53
- `set_shared_config()` (*bitshares.aio.dex.Dex class method*), 55
- `set_shared_config()` (*bit-shares.aio.genesisbalance.GenesisBalance class method*), 57
- `set_shared_config()` (*bit-shares.aio.genesisbalance.GenesisBalances class method*), 58
- `set_shared_config()` (*bitshares.aio.htlc.Htlc class method*), 60
- `set_shared_config()` (*bit-shares.aio.instance.BlockchainInstance class method*), 61
- `set_shared_config()` (*bit-shares.aio.market.Market class method*), 65
- `set_shared_config()` (*bitshares.aio.memo.Memo class method*), 68
- `set_shared_config()` (*bit-shares.aio.message.Message class method*), 69
- `set_shared_config()` (*bit-shares.aio.price.FilledOrder class method*), 71
- `set_shared_config()` (*bitshares.aio.price.Order class method*), 72
- `set_shared_config()` (*bitshares.aio.price.Price class method*), 75
- `set_shared_config()` (*bit-shares.aio.price.PriceFeed class method*), 76
- `set_shared_config()` (*bit-shares.aio.price.UpdateCallOrder class method*), 78
- `set_shared_config()` (*bit-shares.aio.proposal.Proposal class method*), 80
- `set_shared_config()` (*bit-shares.aio.proposal.Proposals class method*),



- 82
- `set_shared_config()` (*bitshares.aio.transactionbuilder.ProposalBuilder class method*), 83
- `set_shared_config()` (*bitshares.aio.transactionbuilder.TransactionBuilder class method*), 85
- `set_shared_config()` (*bitshares.aio.vesting.Vesting class method*), 87
- `set_shared_config()` (*bitshares.aio.wallet.Wallet class method*), 89
- `set_shared_config()` (*bitshares.aio.witness.Witness class method*), 91
- `set_shared_config()` (*bitshares.aio.witness.Witnesses class method*), 93
- `set_shared_config()` (*bitshares.aio.worker.Worker class method*), 95
- `set_shared_config()` (*bitshares.aio.worker.Workers class method*), 97
- `set_shared_config()` (*bitshares.amount.Amount class method*), 103
- `set_shared_config()` (*bitshares.asset.Asset class method*), 107
- `set_shared_config()` (*bitshares.block.Block class method*), 121
- `set_shared_config()` (*bitshares.block.BlockHeader class method*), 123
- `set_shared_config()` (*bitshares.blockchain.Blockchain class method*), 126
- `set_shared_config()` (*bitshares.blockchainobject.BlockchainObject class method*), 128
- `set_shared_config()` (*bitshares.blockchainobject.Object class method*), 129
- `set_shared_config()` (*bitshares.committee.Committee class method*), 131
- `set_shared_config()` (*bitshares.dex.Dex class method*), 134
- `set_shared_config()` (*bitshares.genesisbalance.GenesisBalance class method*), 136
- `set_shared_config()` (*bitshares.genesisbalance.GenesisBalances class method*), 138
- `set_shared_config()` (*bitshares.htlc.Htlc class method*), 140
- `set_shared_config()` (*bitshares.instance.BlockchainInstance class method*), 141
- `set_shared_config()` (*bitshares.market.Market class method*), 145
- `set_shared_config()` (*bitshares.memo.Memo class method*), 148
- `set_shared_config()` (*bitshares.message.Message class method*), 148
- `set_shared_config()` (*bitshares.notify.Notify class method*), 150
- `set_shared_config()` (*bitshares.price.FilledOrder class method*), 152
- `set_shared_config()` (*bitshares.price.Order class method*), 154
- `set_shared_config()` (*bitshares.price.Price class method*), 156
- `set_shared_config()` (*bitshares.price.PriceFeed class method*), 158
- `set_shared_config()` (*bitshares.price.UpdateCallOrder class method*), 159
- `set_shared_config()` (*bitshares.proposal.Proposal class method*), 161
- `set_shared_config()` (*bitshares.proposal.Proposals class method*), 163
- `set_shared_config()` (*bitshares.transactionbuilder.ProposalBuilder class method*), 165
- `set_shared_config()` (*bitshares.transactionbuilder.TransactionBuilder class method*), 167
- `set_shared_config()` (*bitshares.vesting.Vesting class method*), 169
- `set_shared_config()` (*bitshares.wallet.Wallet class method*), 171
- `set_shared_config()` (*bitshares.witness.Witness class method*), 172
- `set_shared_config()` (*bitshares.witness.Witnesses class method*), 175
- `set_shared_config()` (*bitshares.worker.Worker class method*), 176
- `set_shared_config()` (*bitshares.worker.Workers class method*), 178
- `set_shared_config()` (*in module bitshares.aio.instance*), 61
- `set_shared_config()` (*in module bitshares.instance*), 141
- `set_shared_instance()` (*bitshares.account.Account method*), 100
- `set_shared_instance()` (*bit-*

[shares.account.AccountUpdate](#) [method](#)),  
[101](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.account.Account](#) [method](#)), [22](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.account.AccountUpdate](#) [method](#)),  
[23](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.amount.Amount](#) [method](#)), [25](#)  
[set\\_shared\\_instance\(\)](#) ([bitshares.aio.asset.Asset](#)  
[method](#)), [29](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.bitshares.BitShares](#) [method](#)),  
[38](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.block.Block](#) [method](#)), [42](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.block.BlockHeader](#) [method](#)), [44](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.blockchain.Blockchain](#) [method](#)),  
[47](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.blockchainobject.BlockchainObject](#)  
[method](#)), [49](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.blockchainobject.Object](#) [method](#)),  
[51](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.committee.Committee](#) [method](#)),  
[53](#)  
[set\\_shared\\_instance\(\)](#) ([bitshares.aio.dex.Dex](#)  
[method](#)), [55](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.genesisbalance.GenesisBalance](#)  
[method](#)), [57](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.genesisbalance.GenesisBalances](#)  
[method](#)), [58](#)  
[set\\_shared\\_instance\(\)](#) ([bitshares.aio.htlc.Htlc](#)  
[method](#)), [60](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.instance.BlockchainInstance](#)  
[method](#)), [61](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.market.Market](#) [method](#)), [66](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.memo.Memo](#) [method](#)), [68](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.message.Message](#) [method](#)), [69](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.price.FilledOrder](#) [method](#)), [71](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.price.Order](#) [method](#)), [72](#)  
[set\\_shared\\_instance\(\)](#) ([bitshares.aio.price.Price](#)  
[method](#)), [75](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.price.PriceFeed](#) [method](#)), [76](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.price.UpdateCallOrder](#) [method](#)),  
[78](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.proposal.Proposal](#) [method](#)), [80](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.proposal.Proposals](#) [method](#)),  
[82](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.transactionbuilder.ProposalBuilder](#)  
[method](#)), [83](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.transactionbuilder.TransactionBuilder](#)  
[method](#)), [85](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.vesting.Vesting](#) [method](#)), [87](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.wallet.Wallet](#) [method](#)), [89](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.witness.Witness](#) [method](#)), [91](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.witness.Witnesses](#) [method](#)), [93](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.worker.Worker](#) [method](#)), [95](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.aio.worker.Workers](#) [method](#)), [97](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.amount.Amount](#) [method](#)), [103](#)  
[set\\_shared\\_instance\(\)](#) ([bitshares.asset.Asset](#)  
[method](#)), [107](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.bitshares.BitShares](#) [method](#)), [118](#)  
[set\\_shared\\_instance\(\)](#) ([bitshares.block.Block](#)  
[method](#)), [121](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.block.BlockHeader](#) [method](#)), [123](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.blockchain.Blockchain](#) [method](#)), [126](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.blockchainobject.BlockchainObject](#)  
[method](#)), [128](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.blockchainobject.Object](#) [method](#)),  
[130](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.committee.Committee](#) [method](#)), [132](#)  
[set\\_shared\\_instance\(\)](#) ([bitshares.dex.Dex](#)  
[method](#)), [134](#)  
[set\\_shared\\_instance\(\)](#) ([bit-](#)  
[shares.genesisbalance.GenesisBalance](#)  
[method](#)), [137](#)

`set_shared_instance()` (*bitshares.genesisbalance.GenesisBalances method*), 138  
`set_shared_instance()` (*bitshares.htlc.Htlc method*), 140  
`set_shared_instance()` (*bitshares.instance.BlockchainInstance method*), 141  
`set_shared_instance()` (*bitshares.market.Market method*), 145  
`set_shared_instance()` (*bitshares.memo.Memo method*), 148  
`set_shared_instance()` (*bitshares.message.Message method*), 149  
`set_shared_instance()` (*bitshares.notify.Notify method*), 151  
`set_shared_instance()` (*bitshares.price.FilledOrder method*), 152  
`set_shared_instance()` (*bitshares.price.Order method*), 154  
`set_shared_instance()` (*bitshares.price.Price method*), 156  
`set_shared_instance()` (*bitshares.price.PriceFeed method*), 158  
`set_shared_instance()` (*bitshares.price.UpdateCallOrder method*), 159  
`set_shared_instance()` (*bitshares.proposal.Proposal method*), 161  
`set_shared_instance()` (*bitshares.proposal.Proposals method*), 163  
`set_shared_instance()` (*bitshares.transactionbuilder.ProposalBuilder method*), 165  
`set_shared_instance()` (*bitshares.transactionbuilder.TransactionBuilder method*), 167  
`set_shared_instance()` (*bitshares.vesting.Vesting method*), 169  
`set_shared_instance()` (*bitshares.wallet.Wallet method*), 171  
`set_shared_instance()` (*bitshares.witness.Witness method*), 173  
`set_shared_instance()` (*bitshares.witness.Witnesses method*), 175  
`set_shared_instance()` (*bitshares.worker.Worker method*), 176  
`set_shared_instance()` (*bitshares.worker.Workers method*), 178  
`setdefault()` (*bitshares.account.Account method*), 100  
`setdefault()` (*bitshares.account.AccountUpdate method*), 101  
`setdefault()` (*bitshares.aio.account.Account method*), 22  
`setdefault()` (*bitshares.aio.account.AccountUpdate method*), 23  
`setdefault()` (*bitshares.aio.amount.Amount method*), 25  
`setdefault()` (*bitshares.aio.asset.Asset method*), 29  
`setdefault()` (*bitshares.aio.block.Block method*), 42  
`setdefault()` (*bitshares.aio.block.BlockHeader method*), 44  
`setdefault()` (*bitshares.aio.blockchainobject.BlockchainObject method*), 49  
`setdefault()` (*bitshares.aio.blockchainobject.Object method*), 51  
`setdefault()` (*bitshares.aio.committee.Committee method*), 53  
`setdefault()` (*bitshares.aio.genesisbalance.GenesisBalance method*), 57  
`setdefault()` (*bitshares.aio.htlc.Htlc method*), 60  
`setdefault()` (*bitshares.aio.market.Market method*), 66  
`setdefault()` (*bitshares.aio.price.FilledOrder method*), 71  
`setdefault()` (*bitshares.aio.price.Order method*), 72  
`setdefault()` (*bitshares.aio.price.Price method*), 75  
`setdefault()` (*bitshares.aio.price.PriceFeed method*), 76  
`setdefault()` (*bitshares.aio.price.UpdateCallOrder method*), 78  
`setdefault()` (*bitshares.aio.proposal.Proposal method*), 80  
`setdefault()` (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 85  
`setdefault()` (*bitshares.aio.vesting.Vesting method*), 87  
`setdefault()` (*bitshares.aio.witness.Witness method*), 91  
`setdefault()` (*bitshares.aio.worker.Worker method*), 95  
`setdefault()` (*bitshares.amount.Amount method*), 103  
`setdefault()` (*bitshares.asset.Asset method*), 107  
`setdefault()` (*bitshares.block.Block method*), 121  
`setdefault()` (*bitshares.block.BlockHeader method*), 123  
`setdefault()` (*bitshares.blockchainobject.BlockchainObject method*), 128  
`setdefault()` (*bitshares.blockchainobject.Object method*), 130  
`setdefault()` (*bitshares.committee.Committee method*), 132  
`setdefault()` (*bitshares.genesisbalance.GenesisBalance method*), 137  
`setdefault()` (*bitshares.htlc.Htlc method*), 140  
`setdefault()` (*bitshares.market.Market method*), 145

`setdefault()` (*bitshares.price.FilledOrder method*), 152

`setdefault()` (*bitshares.price.Order method*), 154

`setdefault()` (*bitshares.price.Price method*), 156

`setdefault()` (*bitshares.price.PriceFeed method*), 158

`setdefault()` (*bitshares.price.UpdateCallOrder method*), 159

`setdefault()` (*bitshares.proposal.Proposal method*), 161

`setdefault()` (*bitshares.transactionbuilder.TransactionBuilder method*), 167

`setdefault()` (*bitshares.vesting.Vesting method*), 169

`setdefault()` (*bitshares.witness.Witness method*), 173

`setdefault()` (*bitshares.worker.Worker method*), 176

`setdefault()` (*bitsharesbase.objects.AccountCreateExtensions.Buyback\_options method*), 183

`setdefault()` (*bitsharesbase.objects.AccountCreateExtensions.Null\_ext method*), 184

`setdefault()` (*bitsharesbase.objects.AccountOptions method*), 185

`setdefault()` (*bitsharesbase.objects.AssetOptions method*), 186

`setdefault()` (*bitsharesbase.objects.BitAssetOptions method*), 186

`setdefault()` (*bitsharesbase.objects.Memo method*), 187

`setdefault()` (*bitsharesbase.objects.Permission method*), 189

`setdefault()` (*bitsharesbase.objects.Price method*), 190

`setdefault()` (*bitsharesbase.objects.PriceFeed method*), 191

`setdefault()` (*bitsharesbase.operations.Account\_create method*), 192

`setdefault()` (*bitsharesbase.operations.Account\_update method*), 193

`setdefault()` (*bitsharesbase.operations.Account\_upgrade method*), 193

`setdefault()` (*bitsharesbase.operations.Account\_whitelist method*), 194

`setdefault()` (*bitsharesbase.operations.Assert method*), 195

`setdefault()` (*bitsharesbase.operations.Asset\_claim\_fees method*), 196

`setdefault()` (*bitsharesbase.operations.Asset\_claim\_pool method*), 196

`setdefault()` (*bitsharesbase.operations.Asset\_create method*), 197

`setdefault()` (*bitsharesbase.operations.Asset\_fund\_fee\_pool method*), 198

`setdefault()` (*bitsharesbase.operations.Asset\_global\_settle method*), 198

`setdefault()` (*bitsharesbase.operations.Asset\_issue method*), 199

`setdefault()` (*bitsharesbase.operations.Asset\_publish\_feed method*), 200

`setdefault()` (*bitsharesbase.operations.Asset\_reserve method*), 201

`setdefault()` (*bitsharesbase.operations.Asset\_settle method*), 201

`setdefault()` (*bitsharesbase.operations.Asset\_update method*), 202

`setdefault()` (*bitsharesbase.operations.Asset\_update\_bitasset method*), 203

`setdefault()` (*bitsharesbase.operations.Asset\_update\_feed\_producers method*), 203

`setdefault()` (*bitsharesbase.operations.Asset\_update\_issuer method*), 204

`setdefault()` (*bitsharesbase.operations.Balance\_claim method*), 205

`setdefault()` (*bitsharesbase.operations.Bid\_collateral method*), 206

`setdefault()` (*bitsharesbase.operations.Call\_order\_update method*), 206

`setdefault()` (*bitsharesbase.operations.Committee\_member\_create method*), 207

`setdefault()` (*bitsharesbase.operations.Custom method*), 208

`setdefault()` (*bitsharesbase.operations.Htlc\_create method*), 209

`setdefault()` (*bitsharesbase.operations.Htlc\_extend method*), 209

`setdefault()` (*bitsharesbase.operations.Htlc\_redeem method*), 210

`setdefault()` (*bitsharesbase.operations.Limit\_order\_cancel method*),

- 211
- `setdefault()` (*bitshares-base.operations.Limit\_order\_create method*), 211
- `setdefault()` (*bitshares-base.operations.Liquidity\_pool\_create method*), 212
- `setdefault()` (*bitshares-base.operations.Liquidity\_pool\_delete method*), 213
- `setdefault()` (*bitshares-base.operations.Liquidity\_pool\_deposit method*), 214
- `setdefault()` (*bitshares-base.operations.Liquidity\_pool\_exchange method*), 214
- `setdefault()` (*bitshares-base.operations.Liquidity\_pool\_withdraw method*), 215
- `setdefault()` (*bitshares-base.operations.Op\_wrapper method*), 216
- `setdefault()` (*bitshares-base.operations.Override\_transfer method*), 216
- `setdefault()` (*bitshares-base.operations.Proposal\_create method*), 217
- `setdefault()` (*bitshares-base.operations.Proposal\_update method*), 218
- `setdefault()` (*bitshares-base.operations.Ticket\_create\_operation method*), 219
- `setdefault()` (*bitshares-base.operations.Ticket\_update\_operation method*), 219
- `setdefault()` (*bitsharesbase.operations.Transfer method*), 220
- `setdefault()` (*bitshares-base.operations.Vesting\_balance\_withdraw method*), 221
- `setdefault()` (*bitshares-base.operations.Withdraw\_permission\_create method*), 221
- `setdefault()` (*bitshares-base.operations.Witness\_update method*), 222
- `setdefault()` (*bitshares-base.operations.Worker\_create method*), 223
- `setdefault()` (*bitshares-base.signedtransactions.Signed\_Transaction method*), 224
- `setKeys()` (*bitshares.aio.wallet.Wallet method*), 89
- `setKeys()` (*bitshares.wallet.Wallet method*), 171
- `setoptions()` (*bitshares.aio.asset.Asset method*), 29
- `setoptions()` (*bitshares.asset.Asset method*), 108
- `settlements` (*bitshares.aio.asset.Asset attribute*), 29
- `settlements` (*bitshares.asset.Asset attribute*), 108
- `shared_bitshares_instance()` (*in module bitshares.aio.instance*), 61
- `shared_bitshares_instance()` (*in module bitshares.instance*), 141
- `shared_blockchain_instance()` (*bitshares.account.Account method*), 100
- `shared_blockchain_instance()` (*bitshares.account.AccountUpdate method*), 101
- `shared_blockchain_instance()` (*bitshares.aio.account.Account method*), 22
- `shared_blockchain_instance()` (*bitshares.aio.account.AccountUpdate method*), 23
- `shared_blockchain_instance()` (*bitshares.aio.amount.Amount method*), 25
- `shared_blockchain_instance()` (*bitshares.aio.asset.Asset method*), 29
- `shared_blockchain_instance()` (*bitshares.aio.block.Block method*), 42
- `shared_blockchain_instance()` (*bitshares.aio.block.BlockHeader method*), 44
- `shared_blockchain_instance()` (*bitshares.aio.blockchain.Blockchain method*), 47
- `shared_blockchain_instance()` (*bitshares.aio.blockchainobject.BlockchainObject method*), 49
- `shared_blockchain_instance()` (*bitshares.aio.blockchainobject.Object method*), 51
- `shared_blockchain_instance()` (*bitshares.aio.committee.Committee method*), 53
- `shared_blockchain_instance()` (*bitshares.aio.dex.Dex method*), 55
- `shared_blockchain_instance()` (*bitshares.aio.genesisbalance.GenesisBalance method*), 57
- `shared_blockchain_instance()` (*bitshares.aio.genesisbalance.GenesisBalances method*), 59
- `shared_blockchain_instance()` (*bitshares.aio.htlc.Htlc method*), 60
- `shared_blockchain_instance()` (*bitshares.aio.instance.BlockchainInstance method*), 61
- `shared_blockchain_instance()` (*bitshares.aio.market.Market method*), 66



<code>shared_blockchain_instance()</code> <i>shares.aio.memo.Memo method</i> ), 68	(bit-	<code>shared_blockchain_instance()</code> <i>shares.committee.Committee method</i> ), 132	(bit-
<code>shared_blockchain_instance()</code> <i>shares.aio.message.Message method</i> ), 69	(bit-	<code>shared_blockchain_instance()</code> <i>shares.dex.Dex method</i> ), 134	(bit-
<code>shared_blockchain_instance()</code> <i>shares.aio.price.FilledOrder method</i> ), 71	(bit-	<code>shared_blockchain_instance()</code> <i>shares.genesisbalance.GenesisBalance</i>	(bit-
<code>shared_blockchain_instance()</code> <i>shares.aio.price.Order method</i> ), 72	(bit-	<i>method</i> ), 137	
<code>shared_blockchain_instance()</code> <i>shares.aio.price.Price method</i> ), 75	(bit-	<code>shared_blockchain_instance()</code> <i>shares.genesisbalance.GenesisBalances</i>	(bit-
<code>shared_blockchain_instance()</code> <i>shares.aio.price.PriceFeed method</i> ), 76	(bit-	<i>method</i> ), 138	
<code>shared_blockchain_instance()</code> <i>shares.aio.price.UpdateCallOrder</i>	(bit-	<code>shared_blockchain_instance()</code> <i>shares.htlc.Htlc method</i> ), 140	(bit-
<i>method</i> ), 78		<code>shared_blockchain_instance()</code> <i>shares.instance.BlockchainInstance</i>	(bit-
<code>shared_blockchain_instance()</code> <i>shares.aio.proposal.Proposal method</i> ), 80	(bit-	<i>method</i> ), 141	
<code>shared_blockchain_instance()</code> <i>shares.aio.proposal.Proposals</i>	(bit-	<code>shared_blockchain_instance()</code> <i>shares.market.Market method</i> ), 145	(bit-
<i>method</i> ), 82		<code>shared_blockchain_instance()</code> <i>shares.memo.Memo method</i> ), 148	(bit-
<code>shared_blockchain_instance()</code> <i>shares.aio.transactionbuilder.ProposalBuilder</i>	(bit-	<code>shared_blockchain_instance()</code> <i>shares.message.Message method</i> ), 149	(bit-
<i>method</i> ), 83		<code>shared_blockchain_instance()</code> <i>shares.notify.Notify method</i> ), 151	(bit-
<code>shared_blockchain_instance()</code> <i>shares.aio.transactionbuilder.TransactionBuilder</i>	(bit-	<code>shared_blockchain_instance()</code> <i>shares.price.FilledOrder method</i> ), 152	(bit-
<i>method</i> ), 85		<code>shared_blockchain_instance()</code> <i>shares.price.Order method</i> ), 154	(bit-
<code>shared_blockchain_instance()</code> <i>shares.aio.vesting.Vesting method</i> ), 87	(bit-	<code>shared_blockchain_instance()</code> <i>shares.price.Price method</i> ), 156	(bit-
<code>shared_blockchain_instance()</code> <i>shares.aio.wallet.Wallet method</i> ), 89	(bit-	<code>shared_blockchain_instance()</code> <i>shares.price.PriceFeed method</i> ), 158	(bit-
<code>shared_blockchain_instance()</code> <i>shares.aio.witness.Witness method</i> ), 91	(bit-	<code>shared_blockchain_instance()</code> <i>shares.price.UpdateCallOrder</i>	(bit-
<code>shared_blockchain_instance()</code> <i>shares.aio.witness.Witnesses method</i> ), 93	(bit-	<i>method</i> ), 159	
<code>shared_blockchain_instance()</code> <i>shares.aio.worker.Worker method</i> ), 95	(bit-	<code>shared_blockchain_instance()</code> <i>shares.proposal.Proposal method</i> ), 161	(bit-
<code>shared_blockchain_instance()</code> <i>shares.aio.worker.Workers method</i> ), 97	(bit-	<code>shared_blockchain_instance()</code> <i>shares.proposal.Proposals method</i> ), 163	(bit-
<code>shared_blockchain_instance()</code> <i>shares.amount.Amount method</i> ), 104	(bit-	<code>shared_blockchain_instance()</code> <i>shares.transactionbuilder.ProposalBuilder</i>	(bit-
<code>shared_blockchain_instance()</code> <i>shares.asset.Asset method</i> ), 108	(bit-	<i>method</i> ), 165	
<code>shared_blockchain_instance()</code> <i>shares.block.Block method</i> ), 121	(bit-	<code>shared_blockchain_instance()</code> <i>shares.transactionbuilder.TransactionBuilder</i>	(bit-
<code>shared_blockchain_instance()</code> <i>shares.block.BlockHeader method</i> ), 123	(bit-	<i>method</i> ), 167	
<code>shared_blockchain_instance()</code> <i>shares.blockchain.Blockchain method</i> ), 126	(bit-	<code>shared_blockchain_instance()</code> <i>shares.vesting.Vesting method</i> ), 169	(bit-
<code>shared_blockchain_instance()</code> <i>shares.blockchainobject.BlockchainObject</i>	(bit-	<code>shared_blockchain_instance()</code> <i>shares.wallet.Wallet method</i> ), 171	(bit-
<i>method</i> ), 128		<code>shared_blockchain_instance()</code> <i>shares.witness.Witness method</i> ), 173	(bit-
<code>shared_blockchain_instance()</code> <i>shares.blockchainobject.Object</i>	(bit-	<code>shared_blockchain_instance()</code> <i>shares.witness.Witnesses method</i> ), 175	(bit-
<i>method</i> ), 130		<code>shared_blockchain_instance()</code> <i>shares.worker.Worker method</i> ), 177	(bit-

[shared\\_blockchain\\_instance\(\)](#) (*bitshares.worker.Workers method*), 179  
[shared\\_blockchain\\_instance\(\)](#) (*in module bitshares.aio.instance*), 61  
[shared\\_blockchain\\_instance\(\)](#) (*in module bitshares.instance*), 141  
[SharedInstance](#) (*class in bitshares.aio.instance*), 61  
[SharedInstance](#) (*class in bitshares.instance*), 141  
[sign\(\)](#) (*bitshares.aio.bitshares.BitShares method*), 38  
[sign\(\)](#) (*bitshares.aio.message.Message method*), 69  
[sign\(\)](#) (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 85  
[sign\(\)](#) (*bitshares.bitshares.BitShares method*), 118  
[sign\(\)](#) (*bitshares.message.Message method*), 149  
[sign\(\)](#) (*bitshares.transactionbuilder.TransactionBuilder method*), 167  
[sign\(\)](#) (*bitsharesbase.signedtransactions.Signed\_Transaction method*), 224  
[SIGNED\\_MESSAGE\\_ENCAPSULATED](#) (*bitshares.aio.message.Message attribute*), 68  
[SIGNED\\_MESSAGE\\_ENCAPSULATED](#) (*bitshares.message.Message attribute*), 148  
[SIGNED\\_MESSAGE\\_META](#) (*bitshares.aio.message.Message attribute*), 68  
[SIGNED\\_MESSAGE\\_META](#) (*bitshares.message.Message attribute*), 148  
[Signed\\_Transaction](#) (*class in bitsharesbase.signedtransactions*), 223  
[sort\(\)](#) (*bitshares.aio.genesisbalance.GenesisBalances method*), 59  
[sort\(\)](#) (*bitshares.aio.proposal.Proposals method*), 82  
[sort\(\)](#) (*bitshares.aio.witness.Witnesses method*), 93  
[sort\(\)](#) (*bitshares.aio.worker.Workers method*), 97  
[sort\(\)](#) (*bitshares.genesisbalance.GenesisBalances method*), 138  
[sort\(\)](#) (*bitshares.proposal.Proposals method*), 163  
[sort\(\)](#) (*bitshares.witness.Witnesses method*), 175  
[sort\(\)](#) (*bitshares.worker.Workers method*), 179  
[sort\(\)](#) (*bitsharesbase.objects.Operation method*), 189  
[sorted\\_options](#) (*bitsharesbase.objects.AccountCreateExtensions attribute*), 184  
[sorted\\_options](#) (*bitsharesbase.objects.CallOrderExtension attribute*), 187  
[space\\_id](#) (*bitshares.account.Account attribute*), 100  
[space\\_id](#) (*bitshares.aio.account.Account attribute*), 22  
[space\\_id](#) (*bitshares.aio.asset.Asset attribute*), 29  
[space\\_id](#) (*bitshares.aio.block.Block attribute*), 42  
[space\\_id](#) (*bitshares.aio.block.BlockHeader attribute*), 44  
[space\\_id](#) (*bitshares.aio.blockchainobject.BlockchainObject attribute*), 49  
[space\\_id](#) (*bitshares.aio.blockchainobject.Object attribute*), 51  
[space\\_id](#) (*bitshares.aio.committee.Committee attribute*), 53  
[space\\_id](#) (*bitshares.aio.genesisbalance.GenesisBalance attribute*), 57  
[space\\_id](#) (*bitshares.aio.htlc.Htlc attribute*), 60  
[space\\_id](#) (*bitshares.aio.proposal.Proposal attribute*), 80  
[space\\_id](#) (*bitshares.aio.vesting.Vesting attribute*), 87  
[space\\_id](#) (*bitshares.aio.witness.Witness attribute*), 91  
[space\\_id](#) (*bitshares.aio.worker.Worker attribute*), 95  
[space\\_id](#) (*bitshares.asset.Asset attribute*), 108  
[space\\_id](#) (*bitshares.block.Block attribute*), 121  
[space\\_id](#) (*bitshares.block.BlockHeader attribute*), 123  
[space\\_id](#) (*bitshares.blockchainobject.BlockchainObject attribute*), 128  
[space\\_id](#) (*bitshares.blockchainobject.Object attribute*), 130  
[space\\_id](#) (*bitshares.committee.Committee attribute*), 132  
[space\\_id](#) (*bitshares.genesisbalance.GenesisBalance attribute*), 137  
[space\\_id](#) (*bitshares.htlc.Htlc attribute*), 140  
[space\\_id](#) (*bitshares.proposal.Proposal attribute*), 161  
[space\\_id](#) (*bitshares.vesting.Vesting attribute*), 169  
[space\\_id](#) (*bitshares.witness.Witness attribute*), 173  
[space\\_id](#) (*bitshares.worker.Worker attribute*), 177  
[SpecialAuthority](#) (*class in bitsharesbase.objects*), 191  
[store\(\)](#) (*bitshares.account.Account method*), 100  
[store\(\)](#) (*bitshares.aio.account.Account method*), 22  
[store\(\)](#) (*bitshares.aio.asset.Asset method*), 30  
[store\(\)](#) (*bitshares.aio.block.Block method*), 42  
[store\(\)](#) (*bitshares.aio.block.BlockHeader method*), 44  
[store\(\)](#) (*bitshares.aio.blockchainobject.BlockchainObject method*), 49  
[store\(\)](#) (*bitshares.aio.blockchainobject.Object method*), 51  
[store\(\)](#) (*bitshares.aio.committee.Committee method*), 53  
[store\(\)](#) (*bitshares.aio.genesisbalance.GenesisBalance method*), 57  
[store\(\)](#) (*bitshares.aio.htlc.Htlc method*), 60  
[store\(\)](#) (*bitshares.aio.proposal.Proposal method*), 80  
[store\(\)](#) (*bitshares.aio.proposal.Proposals method*), 82  
[store\(\)](#) (*bitshares.aio.vesting.Vesting method*), 87  
[store\(\)](#) (*bitshares.aio.witness.Witness method*), 91  
[store\(\)](#) (*bitshares.aio.witness.Witnesses method*), 93  
[store\(\)](#) (*bitshares.aio.worker.Worker method*), 95  
[store\(\)](#) (*bitshares.aio.worker.Workers method*), 97  
[store\(\)](#) (*bitshares.asset.Asset method*), 108  
[store\(\)](#) (*bitshares.block.Block method*), 121  
[store\(\)](#) (*bitshares.block.BlockHeader method*), 123

[store\(\)](#) (*bitshares.blockchainobject.BlockchainObject method*), 128  
[store\(\)](#) (*bitshares.blockchainobject.Object method*), 130  
[store\(\)](#) (*bitshares.committee.Committee method*), 132  
[store\(\)](#) (*bitshares.genesisbalance.GenesisBalance method*), 137  
[store\(\)](#) (*bitshares.htlc.Htlc method*), 140  
[store\(\)](#) (*bitshares.proposal.Proposal method*), 161  
[store\(\)](#) (*bitshares.proposal.Proposals method*), 163  
[store\(\)](#) (*bitshares.vesting.Vesting method*), 169  
[store\(\)](#) (*bitshares.witness.Witness method*), 173  
[store\(\)](#) (*bitshares.witness.Witnesses method*), 175  
[store\(\)](#) (*bitshares.worker.Worker method*), 177  
[store\(\)](#) (*bitshares.worker.Workers method*), 179  
[stream\(\)](#) (*bitshares.aio.blockchain.Blockchain method*), 47  
[stream\(\)](#) (*bitshares.blockchain.Blockchain method*), 126  
[subscribe\\_to\\_accounts\(\)](#) (*bitshares.aio.bitshares.BitShares method*), 39  
[subscribe\\_to\\_blocks\(\)](#) (*bitshares.aio.bitshares.BitShares method*), 39  
[subscribe\\_to\\_market\(\)](#) (*bitshares.aio.bitshares.BitShares method*), 39  
[subscribe\\_to\\_pending\\_transactions\(\)](#) (*bitshares.aio.bitshares.BitShares method*), 39  
[suggest\(\)](#) (*bitsharesbase.account.BrainKey static method*), 180  
[supported\\_formats](#) (*bitshares.aio.message.Message attribute*), 69  
[supported\\_formats](#) (*bitshares.message.Message attribute*), 149  
[symbol](#) (*bitshares.aio.amount.Amount attribute*), 25  
[symbol](#) (*bitshares.aio.asset.Asset attribute*), 30  
[symbol](#) (*bitshares.amount.Amount attribute*), 104  
[symbol](#) (*bitshares.asset.Asset attribute*), 108  
[symbols\(\)](#) (*bitshares.aio.price.FilledOrder method*), 71  
[symbols\(\)](#) (*bitshares.aio.price.Order method*), 73  
[symbols\(\)](#) (*bitshares.aio.price.Price method*), 75  
[symbols\(\)](#) (*bitshares.aio.price.UpdateCallOrder method*), 78  
[symbols\(\)](#) (*bitshares.price.FilledOrder method*), 152  
[symbols\(\)](#) (*bitshares.price.Order method*), 154  
[symbols\(\)](#) (*bitshares.price.Price method*), 156  
[symbols\(\)](#) (*bitshares.price.UpdateCallOrder method*), 159  
[targetCollateralRatio\(\)](#) (*bitshares-*  
*base.objects.CallOrderExtension method*), 187  
[test\\_permissions\(\)](#) (*in module bitshares-base.asset\_permissions*), 182  
[test\\_valid\\_objectid\(\)](#) (*bitshares.account.Account method*), 100  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.account.Account method*), 22  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.asset.Asset method*), 30  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.block.Block method*), 42  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.block.BlockHeader method*), 44  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.blockchainobject.BlockchainObject method*), 49  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.blockchainobject.Object method*), 51  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.committee.Committee method*), 53  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.genesisbalance.GenesisBalance method*), 57  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.htlc.Htlc method*), 60  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.proposal.Proposal method*), 80  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.vesting.Vesting method*), 87  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.witness.Witness method*), 91  
[test\\_valid\\_objectid\(\)](#) (*bitshares.aio.worker.Worker method*), 95  
[test\\_valid\\_objectid\(\)](#) (*bitshares.asset.Asset method*), 108  
[test\\_valid\\_objectid\(\)](#) (*bitshares.block.Block method*), 121  
[test\\_valid\\_objectid\(\)](#) (*bitshares.block.BlockHeader method*), 123  
[test\\_valid\\_objectid\(\)](#) (*bitshares.blockchainobject.BlockchainObject method*), 128  
[test\\_valid\\_objectid\(\)](#) (*bitshares.blockchainobject.Object method*), 130  
[test\\_valid\\_objectid\(\)](#) (*bitshares.committee.Committee method*), 132  
[test\\_valid\\_objectid\(\)](#) (*bitshares.genesisbalance.GenesisBalance method*), 137  
[test\\_valid\\_objectid\(\)](#) (*bitshares.htlc.Htlc*

## T



- method*), 140
- `test_valid_objectid()` (*bitshares.proposal.Proposal method*), 161
- `test_valid_objectid()` (*bitshares.vesting.Vesting method*), 169
- `test_valid_objectid()` (*bitshares.witness.Witness method*), 173
- `test_valid_objectid()` (*bitshares.worker.Worker method*), 177
- `testid()` (*bitshares.account.Account method*), 100
- `testid()` (*bitshares.aio.account.Account method*), 22
- `testid()` (*bitshares.aio.asset.Asset method*), 30
- `testid()` (*bitshares.aio.block.Block method*), 42
- `testid()` (*bitshares.aio.block.BlockHeader method*), 44
- `testid()` (*bitshares.aio.blockchainobject.BlockchainObject method*), 49
- `testid()` (*bitshares.aio.blockchainobject.Object method*), 51
- `testid()` (*bitshares.aio.committee.Committee method*), 53
- `testid()` (*bitshares.aio.genesisbalance.GenesisBalance method*), 57
- `testid()` (*bitshares.aio.htlc.Htlc method*), 60
- `testid()` (*bitshares.aio.proposal.Proposal method*), 80
- `testid()` (*bitshares.aio.vesting.Vesting method*), 87
- `testid()` (*bitshares.aio.witness.Witness method*), 91
- `testid()` (*bitshares.aio.worker.Worker method*), 95
- `testid()` (*bitshares.asset.Asset method*), 108
- `testid()` (*bitshares.block.Block method*), 121
- `testid()` (*bitshares.block.BlockHeader method*), 123
- `testid()` (*bitshares.blockchainobject.BlockchainObject method*), 128
- `testid()` (*bitshares.blockchainobject.Object method*), 130
- `testid()` (*bitshares.committee.Committee method*), 132
- `testid()` (*bitshares.genesisbalance.GenesisBalance method*), 137
- `testid()` (*bitshares.htlc.Htlc method*), 140
- `testid()` (*bitshares.proposal.Proposal method*), 161
- `testid()` (*bitshares.vesting.Vesting method*), 169
- `testid()` (*bitshares.witness.Witness method*), 173
- `testid()` (*bitshares.worker.Worker method*), 177
- `ticker()` (*bitshares.aio.market.Market method*), 66
- `ticker()` (*bitshares.market.Market method*), 145
- `Ticket_create_operation` (class in *bitshares-base.operations*), 218
- `Ticket_update_operation` (class in *bitshares-base.operations*), 219
- `time()` (*bitshares.aio.block.Block method*), 43
- `time()` (*bitshares.aio.block.BlockHeader method*), 44
- `time()` (*bitshares.block.Block method*), 122
- `time()` (*bitshares.block.BlockHeader method*), 123
- `to_buy` (*bitshares.price.Order attribute*), 154
- `todict()` (in module *bitsharesbase.asset\_permissions*), 182
- `toint()` (in module *bitsharesbase.asset\_permissions*), 182
- `toJson()` (*bitshares-base.objects.AccountCreateExtensions.Buyback\_options method*), 183
- `toJson()` (*bitshares-base.objects.AccountCreateExtensions.Null\_ext method*), 184
- `toJson()` (*bitsharesbase.objects.AccountOptions method*), 185
- `toJson()` (*bitsharesbase.objects.AssetOptions method*), 186
- `toJson()` (*bitsharesbase.objects.BitAssetOptions method*), 186
- `toJson()` (*bitsharesbase.objects.Memo method*), 187
- `toJson()` (*bitsharesbase.objects.Operation method*), 189
- `toJson()` (*bitsharesbase.objects.Permission method*), 189
- `toJson()` (*bitsharesbase.objects.Price method*), 190
- `toJson()` (*bitsharesbase.objects.PriceFeed method*), 191
- `toJson()` (*bitsharesbase.operations.Account\_create method*), 192
- `toJson()` (*bitsharesbase.operations.Account\_update method*), 193
- `toJson()` (*bitsharesbase.operations.Account\_upgrade method*), 193
- `toJson()` (*bitsharesbase.operations.Account\_whitelist method*), 194
- `toJson()` (*bitsharesbase.operations.Assert method*), 195
- `toJson()` (*bitsharesbase.operations.Asset\_claim\_fees method*), 196
- `toJson()` (*bitsharesbase.operations.Asset\_claim\_pool method*), 196
- `toJson()` (*bitsharesbase.operations.Asset\_create method*), 197
- `toJson()` (*bitshares-base.operations.Asset\_fund\_fee\_pool method*), 198
- `toJson()` (*bitshares-base.operations.Asset\_global\_settle method*), 198
- `toJson()` (*bitsharesbase.operations.Asset\_issue method*), 199
- `toJson()` (*bitshares-base.operations.Asset\_publish\_feed method*), 200
- `toJson()` (*bitsharesbase.operations.Asset\_reserve*

- method), 201
- toJson() (bitsharesbase.operations.Asset\_settle method), 201
- toJson() (bitsharesbase.operations.Asset\_update method), 202
- toJson() (bitsharesbase.operations.Asset\_update\_bitasset method), 203
- toJson() (bitsharesbase.operations.Asset\_update\_feed\_producers method), 203
- toJson() (bitsharesbase.operations.Asset\_update\_issuer method), 204
- toJson() (bitsharesbase.operations.Balance\_claim method), 205
- toJson() (bitsharesbase.operations.Bid\_collateral method), 206
- toJson() (bitsharesbase.operations.Call\_order\_update method), 206
- toJson() (bitsharesbase.operations.Committee\_member\_create method), 207
- toJson() (bitsharesbase.operations.Custom method), 208
- toJson() (bitsharesbase.operations.Htlc\_create method), 209
- toJson() (bitsharesbase.operations.Htlc\_extend method), 209
- toJson() (bitsharesbase.operations.Htlc\_redeem method), 210
- toJson() (bitsharesbase.operations.Limit\_order\_cancel method), 211
- toJson() (bitsharesbase.operations.Limit\_order\_create method), 212
- toJson() (bitsharesbase.operations.Liquidity\_pool\_create method), 212
- toJson() (bitsharesbase.operations.Liquidity\_pool\_delete method), 213
- toJson() (bitsharesbase.operations.Liquidity\_pool\_deposit method), 214
- toJson() (bitsharesbase.operations.Liquidity\_pool\_exchange method), 214
- toJson() (bitsharesbase.operations.Liquidity\_pool\_withdraw method), 215
- toJson() (bitsharesbase.operations.Op\_wrapper method), 216
- toJson() (bitsharesbase.operations.Override\_transfer method), 217
- toJson() (bitsharesbase.operations.Proposal\_create method), 217
- toJson() (bitsharesbase.operations.Proposal\_update method), 218
- toJson() (bitsharesbase.operations.Ticket\_create\_operation method), 219
- toJson() (bitsharesbase.operations.Ticket\_update\_operation method), 219
- toJson() (bitsharesbase.operations.Transfer method), 220
- toJson() (bitsharesbase.operations.Vesting\_balance\_withdraw method), 221
- toJson() (bitsharesbase.operations.Withdraw\_permission\_create method), 222
- toJson() (bitsharesbase.operations.Witness\_update method), 222
- toJson() (bitsharesbase.operations.Worker\_create method), 223
- toJson() (bitsharesbase.signedtransactions.Signed\_Transaction method), 225
- trades() (bitshares.aio.market.Market method), 66
- trades() (bitshares.market.Market method), 146
- TransactionBuilder (class in bitshares.aio.transactionbuilder), 83
- TransactionBuilder (class in bitshares.transactionbuilder), 165
- Transfer (class in bitsharesbase.operations), 220
- transfer() (bitshares.aio.bitshares.BitShares method), 39
- transfer() (bitshares.bitshares.BitShares method), 118
- tuple() (bitshares.aio.amount.Amount method), 25
- tuple() (bitshares.amount.Amount method), 104
- tx() (bitshares.aio.bitshares.BitShares method), 39
- tx() (bitshares.bitshares.BitShares method), 118
- txbuffer (bitshares.aio.bitshares.BitShares attribute), 39
- txbuffer (bitshares.bitshares.BitShares attribute), 118
- type\_id (bitshares.account.Account attribute), 100
- type\_id (bitshares.aio.account.Account attribute), 22
- type\_id (bitshares.aio.asset.Asset attribute), 30
- type\_id (bitshares.aio.block.Block attribute), 43
- type\_id (bitshares.aio.block.BlockHeader attribute), 44
- type\_id (bitshares.aio.blockchainobject.BlockchainObject attribute), 49

- `type_id` (`bitshares.aio.blockchainobject.Object` attribute), 51
- `type_id` (`bitshares.aio.committee.Committee` attribute), 53
- `type_id` (`bitshares.aio.genesisbalance.GenesisBalance` attribute), 57
- `type_id` (`bitshares.aio.htlc.Htlc` attribute), 60
- `type_id` (`bitshares.aio.proposal.Proposal` attribute), 80
- `type_id` (`bitshares.aio.vesting.Vesting` attribute), 87
- `type_id` (`bitshares.aio.witness.Witness` attribute), 91
- `type_id` (`bitshares.aio.worker.Worker` attribute), 95
- `type_id` (`bitshares.asset.Asset` attribute), 108
- `type_id` (`bitshares.block.Block` attribute), 122
- `type_id` (`bitshares.block.BlockHeader` attribute), 123
- `type_id` (`bitshares.blockchainobject.BlockchainObject` attribute), 128
- `type_id` (`bitshares.blockchainobject.Object` attribute), 130
- `type_id` (`bitshares.committee.Committee` attribute), 132
- `type_id` (`bitshares.genesisbalance.GenesisBalance` attribute), 137
- `type_id` (`bitshares.htlc.Htlc` attribute), 140
- `type_id` (`bitshares.proposal.Proposal` attribute), 161
- `type_id` (`bitshares.vesting.Vesting` attribute), 169
- `type_id` (`bitshares.witness.Witness` attribute), 173
- `type_id` (`bitshares.worker.Worker` attribute), 177
- `type_ids` (`bitshares.account.Account` attribute), 100
- `type_ids` (`bitshares.aio.account.Account` attribute), 22
- `type_ids` (`bitshares.aio.asset.Asset` attribute), 30
- `type_ids` (`bitshares.aio.block.Block` attribute), 43
- `type_ids` (`bitshares.aio.block.BlockHeader` attribute), 44
- `type_ids` (`bitshares.aio.blockchainobject.BlockchainObject` attribute), 49
- `type_ids` (`bitshares.aio.blockchainobject.Object` attribute), 51
- `type_ids` (`bitshares.aio.committee.Committee` attribute), 53
- `type_ids` (`bitshares.aio.genesisbalance.GenesisBalance` attribute), 57
- `type_ids` (`bitshares.aio.htlc.Htlc` attribute), 60
- `type_ids` (`bitshares.aio.proposal.Proposal` attribute), 80
- `type_ids` (`bitshares.aio.vesting.Vesting` attribute), 87
- `type_ids` (`bitshares.aio.witness.Witness` attribute), 91
- `type_ids` (`bitshares.aio.worker.Worker` attribute), 95
- `type_ids` (`bitshares.asset.Asset` attribute), 108
- `type_ids` (`bitshares.block.Block` attribute), 122
- `type_ids` (`bitshares.block.BlockHeader` attribute), 123
- `type_ids` (`bitshares.blockchainobject.BlockchainObject` attribute), 128
- `type_ids` (`bitshares.blockchainobject.Object` attribute), 130
- `type_ids` (`bitshares.committee.Committee` attribute), 132
- `type_ids` (`bitshares.genesisbalance.GenesisBalance` attribute), 137
- `type_ids` (`bitshares.htlc.Htlc` attribute), 140
- `type_ids` (`bitshares.proposal.Proposal` attribute), 161
- `type_ids` (`bitshares.vesting.Vesting` attribute), 169
- `type_ids` (`bitshares.witness.Witness` attribute), 173
- `type_ids` (`bitshares.worker.Worker` attribute), 177
- `uncompressed` (`bitsharesbase.account.PrivateKey` attribute), 181
- `unCompressed()` (`bitsharesbase.account.PublicKey` method), 182
- `uncompressed()` (`bitsharesbase.account.PublicKey` method), 182
- `unlock()` (`bitshares.aio.bitshares.BitShares` method), 39
- `unlock()` (`bitshares.aio.wallet.Wallet` method), 89
- `unlock()` (`bitshares.bitshares.BitShares` method), 118
- `unlock()` (`bitshares.wallet.Wallet` method), 171
- `unlock_wallet()` (`bitshares.aio.memo.Memo` method), 68
- `unlock_wallet()` (`bitshares.memo.Memo` method), 148
- `unlocked()` (`bitshares.aio.wallet.Wallet` method), 89
- `unlocked()` (`bitshares.wallet.Wallet` method), 171
- `unset_proxy()` (`bitshares.aio.bitshares.BitShares` method), 39
- `unset_proxy()` (`bitshares.bitshares.BitShares` method), 118
- `update()` (`bitshares.account.Account` method), 100
- `update()` (`bitshares.account.AccountUpdate` method), 101
- `update()` (`bitshares.aio.account.Account` method), 22
- `update()` (`bitshares.aio.account.AccountUpdate` method), 23
- `update()` (`bitshares.aio.amount.Amount` method), 25
- `update()` (`bitshares.aio.asset.Asset` method), 30
- `update()` (`bitshares.aio.block.Block` method), 43
- `update()` (`bitshares.aio.block.BlockHeader` method), 45
- `update()` (`bitshares.aio.blockchainobject.BlockchainObject` method), 49
- `update()` (`bitshares.aio.blockchainobject.Object` method), 51
- `update()` (`bitshares.aio.committee.Committee` method), 53
- `update()` (`bitshares.aio.genesisbalance.GenesisBalance` method), 57
- `update()` (`bitshares.aio.htlc.Htlc` method), 61
- `update()` (`bitshares.aio.market.Market` method), 66
- `update()` (`bitshares.aio.price.FilledOrder` method), 71

`update()` (*bitshares.aio.price.Order method*), 73  
`update()` (*bitshares.aio.price.Price method*), 75  
`update()` (*bitshares.aio.price.PriceFeed method*), 76  
`update()` (*bitshares.aio.price.UpdateCallOrder method*), 78  
`update()` (*bitshares.aio.proposal.Proposal method*), 80  
`update()` (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 85  
`update()` (*bitshares.aio.vesting.Vesting method*), 87  
`update()` (*bitshares.aio.witness.Witness method*), 91  
`update()` (*bitshares.aio.worker.Worker method*), 95  
`update()` (*bitshares.amount.Amount method*), 104  
`update()` (*bitshares.asset.Asset method*), 108  
`update()` (*bitshares.block.Block method*), 122  
`update()` (*bitshares.block.BlockHeader method*), 123  
`update()` (*bitshares.blockchainobject.BlockchainObject method*), 128  
`update()` (*bitshares.blockchainobject.Object method*), 130  
`update()` (*bitshares.committee.Committee method*), 132  
`update()` (*bitshares.genesisbalance.GenesisBalance method*), 137  
`update()` (*bitshares.htlc.Htlc method*), 140  
`update()` (*bitshares.market.Market method*), 146  
`update()` (*bitshares.price.FilledOrder method*), 152  
`update()` (*bitshares.price.Order method*), 154  
`update()` (*bitshares.price.Price method*), 156  
`update()` (*bitshares.price.PriceFeed method*), 158  
`update()` (*bitshares.price.UpdateCallOrder method*), 159  
`update()` (*bitshares.proposal.Proposal method*), 161  
`update()` (*bitshares.transactionbuilder.TransactionBuilder method*), 167  
`update()` (*bitshares.vesting.Vesting method*), 169  
`update()` (*bitshares.witness.Witness method*), 173  
`update()` (*bitshares.worker.Worker method*), 177  
`update()` (*bitsharesbase.objects.AccountCreateExtensions.Buyback\_options method*), 183  
`update()` (*bitsharesbase.objects.AccountCreateExtensions.Null\_ext method*), 184  
`update()` (*bitsharesbase.objects.AccountOptions method*), 185  
`update()` (*bitsharesbase.objects.AssetOptions method*), 186  
`update()` (*bitsharesbase.objects.BitAssetOptions method*), 186  
`update()` (*bitsharesbase.objects.Memo method*), 187  
`update()` (*bitsharesbase.objects.Permission method*), 189  
`update()` (*bitsharesbase.objects.Price method*), 190  
`update()` (*bitsharesbase.objects.PriceFeed method*), 191  
`update()` (*bitsharesbase.operations.Account\_create method*), 192  
`update()` (*bitsharesbase.operations.Account\_update method*), 193  
`update()` (*bitsharesbase.operations.Account\_upgrade method*), 193  
`update()` (*bitsharesbase.operations.Account\_whitelist method*), 194  
`update()` (*bitsharesbase.operations.Assert method*), 195  
`update()` (*bitsharesbase.operations.Asset\_claim\_fees method*), 196  
`update()` (*bitsharesbase.operations.Asset\_claim\_pool method*), 196  
`update()` (*bitsharesbase.operations.Asset\_create method*), 197  
`update()` (*bitsharesbase.operations.Asset\_fund\_fee\_pool method*), 198  
`update()` (*bitsharesbase.operations.Asset\_global\_settle method*), 199  
`update()` (*bitsharesbase.operations.Asset\_issue method*), 199  
`update()` (*bitsharesbase.operations.Asset\_publish\_feed method*), 200  
`update()` (*bitsharesbase.operations.Asset\_reserve method*), 201  
`update()` (*bitsharesbase.operations.Asset\_settle method*), 201  
`update()` (*bitsharesbase.operations.Asset\_update method*), 202  
`update()` (*bitsharesbase.operations.Asset\_update\_bitasset method*), 203  
`update()` (*bitsharesbase.operations.Asset\_update\_feed\_producers method*), 204  
`update()` (*bitsharesbase.operations.Asset\_update\_issuer method*), 204  
`update()` (*bitsharesbase.operations.Balance\_claim method*), 205  
`update()` (*bitsharesbase.operations.Bid\_collateral method*), 206  
`update()` (*bitsharesbase.operations.Call\_order\_update method*), 206  
`update()` (*bitsharesbase.operations.Committee\_member\_create method*), 207

- update () (*bitsharesbase.operations.Custom method*), 208
- update () (*bitsharesbase.operations.Htlc\_create method*), 209
- update () (*bitsharesbase.operations.Htlc\_extend method*), 209
- update () (*bitsharesbase.operations.Htlc\_redeem method*), 210
- update () (*bitsharesbase.operations.Limit\_order\_cancel method*), 211
- update () (*bitsharesbase.operations.Limit\_order\_create method*), 212
- update () (*bitsharesbase.operations.Liquidity\_pool\_create method*), 212
- update () (*bitsharesbase.operations.Liquidity\_pool\_delete method*), 213
- update () (*bitsharesbase.operations.Liquidity\_pool\_deposit method*), 214
- update () (*bitsharesbase.operations.Liquidity\_pool\_exchange method*), 214
- update () (*bitsharesbase.operations.Liquidity\_pool\_withdraw method*), 215
- update () (*bitsharesbase.operations.Op\_wrapper method*), 216
- update () (*bitsharesbase.operations.Override\_transfer method*), 217
- update () (*bitsharesbase.operations.Proposal\_create method*), 217
- update () (*bitsharesbase.operations.Proposal\_update method*), 218
- update () (*bitsharesbase.operations.Ticket\_create\_operation method*), 219
- update () (*bitsharesbase.operations.Ticket\_update\_operation method*), 219
- update () (*bitsharesbase.operations.Transfer method*), 220
- update () (*bitsharesbase.operations.Vesting\_balance\_withdraw method*), 221
- update () (*bitsharesbase.operations.Withdraw\_permission\_create method*), 222
- update () (*bitsharesbase.operations.Witness\_update method*), 222
- update () (*bitsharesbase.operations.Worker\_create method*), 223
- update () (*bitsharesbase.signedtransactions.Signed\_Transaction method*), 225
- update\_cer () (*bitshares.aio.asset.Asset method*), 30
- update\_cer () (*bitshares.aio.bitshares.BitShares method*), 39
- update\_cer () (*bitshares.asset.Asset method*), 108
- update\_cer () (*bitshares.bitshares.BitShares method*), 118
- update\_chain\_parameters () (*bitshares.aio.blockchain.Blockchain method*), 47
- update\_chain\_parameters () (*bitshares.blockchain.Blockchain method*), 126
- update\_feed\_producers () (*bitshares.aio.asset.Asset method*), 30
- update\_feed\_producers () (*bitshares.asset.Asset method*), 108
- update\_memo\_key () (*bitshares.aio.bitshares.BitShares method*), 40
- update\_memo\_key () (*bitshares.bitshares.BitShares method*), 118
- update\_voting\_ticket () (*bitshares.aio.bitshares.BitShares method*), 40
- update\_voting\_ticket () (*bitshares.bitshares.BitShares method*), 119
- update\_witness () (*bitshares.aio.bitshares.BitShares method*), 40
- update\_witness () (*bitshares.bitshares.BitShares method*), 119
- UpdateCallOrder (*class in bitshares.aio.price*), 76
- UpdateCallOrder (*class in bitshares.price*), 158
- upgrade () (*bitshares.account.Account method*), 100
- upgrade () (*bitshares.aio.account.Account method*), 22
- upgrade\_account () (*bitshares.aio.bitshares.BitShares method*), 40
- upgrade\_account () (*bitshares.bitshares.BitShares method*), 119
- ## V
- valid\_exceptions (*bitshares.aio.message.Message attribute*), 69
- valid\_exceptions (*bitshares.message.Message attribute*), 149
- values () (*bitshares.account.Account method*), 100
- values () (*bitshares.account.AccountUpdate method*), 102
- values () (*bitshares.aio.account.Account method*), 22



`values()` (*bitshares.aio.account.AccountUpdate method*), 23

`values()` (*bitshares.aio.amount.Amount method*), 26

`values()` (*bitshares.aio.asset.Asset method*), 30

`values()` (*bitshares.aio.block.Block method*), 43

`values()` (*bitshares.aio.block.BlockHeader method*), 45

`values()` (*bitshares.aio.blockchainobject.BlockchainObject method*), 49

`values()` (*bitshares.aio.blockchainobject.Object method*), 51

`values()` (*bitshares.aio.committee.Committee method*), 53

`values()` (*bitshares.aio.genesisbalance.GenesisBalance method*), 57

`values()` (*bitshares.aio.htlc.Htlc method*), 61

`values()` (*bitshares.aio.market.Market method*), 66

`values()` (*bitshares.aio.price.FilledOrder method*), 71

`values()` (*bitshares.aio.price.Order method*), 73

`values()` (*bitshares.aio.price.Price method*), 75

`values()` (*bitshares.aio.price.PriceFeed method*), 76

`values()` (*bitshares.aio.price.UpdateCallOrder method*), 78

`values()` (*bitshares.aio.proposal.Proposal method*), 80

`values()` (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 85

`values()` (*bitshares.aio.vesting.Vesting method*), 87

`values()` (*bitshares.aio.witness.Witness method*), 91

`values()` (*bitshares.aio.worker.Worker method*), 95

`values()` (*bitshares.amount.Amount method*), 104

`values()` (*bitshares.asset.Asset method*), 109

`values()` (*bitshares.block.Block method*), 122

`values()` (*bitshares.block.BlockHeader method*), 124

`values()` (*bitshares.blockchainobject.BlockchainObject method*), 128

`values()` (*bitshares.blockchainobject.Object method*), 130

`values()` (*bitshares.committee.Committee method*), 132

`values()` (*bitshares.genesisbalance.GenesisBalance method*), 137

`values()` (*bitshares.htlc.Htlc method*), 140

`values()` (*bitshares.market.Market method*), 146

`values()` (*bitshares.price.FilledOrder method*), 152

`values()` (*bitshares.price.Order method*), 154

`values()` (*bitshares.price.Price method*), 156

`values()` (*bitshares.price.PriceFeed method*), 158

`values()` (*bitshares.price.UpdateCallOrder method*), 159

`values()` (*bitshares.proposal.Proposal method*), 162

`values()` (*bitshares.transactionbuilder.TransactionBuilder method*), 167

`values()` (*bitshares.vesting.Vesting method*), 169

`values()` (*bitshares.witness.Witness method*), 173

`values()` (*bitshares.worker.Worker method*), 177

`values()` (*bitshares-base.objects.AccountCreateExtensions.Buyback\_options method*), 184

`values()` (*bitshares-base.objects.AccountCreateExtensions.Null\_ext method*), 184

`values()` (*bitsharesbase.objects.AccountOptions method*), 185

`values()` (*bitsharesbase.objects.AssetOptions method*), 186

`values()` (*bitsharesbase.objects.BitAssetOptions method*), 187

`values()` (*bitsharesbase.objects.Memo method*), 187

`values()` (*bitsharesbase.objects.Permission method*), 189

`values()` (*bitsharesbase.objects.Price method*), 190

`values()` (*bitsharesbase.objects.PriceFeed method*), 191

`values()` (*bitsharesbase.operations.Account\_create method*), 192

`values()` (*bitsharesbase.operations.Account\_update method*), 193

`values()` (*bitsharesbase.operations.Account\_upgrade method*), 193

`values()` (*bitsharesbase.operations.Account\_whitelist method*), 194

`values()` (*bitsharesbase.operations.Assert method*), 195

`values()` (*bitsharesbase.operations.Asset\_claim\_fees method*), 196

`values()` (*bitsharesbase.operations.Asset\_claim\_pool method*), 196

`values()` (*bitsharesbase.operations.Asset\_create method*), 197

`values()` (*bitshares-base.operations.Asset\_fund\_fee\_pool method*), 198

`values()` (*bitshares-base.operations.Asset\_global\_settle method*), 199

`values()` (*bitsharesbase.operations.Asset\_issue method*), 199

`values()` (*bitshares-base.operations.Asset\_publish\_feed method*), 200

`values()` (*bitsharesbase.operations.Asset\_reserve method*), 201

`values()` (*bitsharesbase.operations.Asset\_settle method*), 201

`values()` (*bitsharesbase.operations.Asset\_update method*), 202

`values()` (*bitshares-*

- base.operations.Asset\_update\_bitasset method*), 203
- values()* (*bitshares-base.operations.Asset\_update\_feed\_producers method*), 204
- values()* (*bitshares-base.operations.Asset\_update\_issuer method*), 204
- values()* (*bitsharesbase.operations.Balance\_claim method*), 205
- values()* (*bitsharesbase.operations.Bid\_collateral method*), 206
- values()* (*bitshares-base.operations.Call\_order\_update method*), 206
- values()* (*bitshares-base.operations.Committee\_member\_create method*), 207
- values()* (*bitsharesbase.operations.Custom method*), 208
- values()* (*bitsharesbase.operations.Htlc\_create method*), 209
- values()* (*bitsharesbase.operations.Htlc\_extend method*), 209
- values()* (*bitsharesbase.operations.Htlc\_redeem method*), 210
- values()* (*bitshares-base.operations.Limit\_order\_cancel method*), 211
- values()* (*bitshares-base.operations.Limit\_order\_create method*), 212
- values()* (*bitshares-base.operations.Liquidity\_pool\_create method*), 212
- values()* (*bitshares-base.operations.Liquidity\_pool\_delete method*), 213
- values()* (*bitshares-base.operations.Liquidity\_pool\_deposit method*), 214
- values()* (*bitshares-base.operations.Liquidity\_pool\_exchange method*), 214
- values()* (*bitshares-base.operations.Liquidity\_pool\_withdraw method*), 215
- values()* (*bitsharesbase.operations.Op\_wrapper method*), 216
- values()* (*bitsharesbase.operations.Override\_transfer method*), 217
- values()* (*bitsharesbase.operations.Proposal\_create method*), 217
- values()* (*bitsharesbase.operations.Proposal\_update method*), 218
- values()* (*bitshares-base.operations.Ticket\_create\_operation method*), 219
- values()* (*bitshares-base.operations.Ticket\_update\_operation method*), 219
- values()* (*bitsharesbase.operations.Transfer method*), 220
- values()* (*bitshares-base.operations.Vesting\_balance\_withdraw method*), 221
- values()* (*bitshares-base.operations.Withdraw\_permission\_create method*), 222
- values()* (*bitsharesbase.operations.Witness\_update method*), 222
- values()* (*bitsharesbase.operations.Worker\_create method*), 223
- values()* (*bitshares-base.signedtransactions.Signed\_Transaction method*), 225
- verify()* (*bitshares.aio.message.Message method*), 69
- verify()* (*bitshares.message.Message method*), 149
- verify()* (*bitshares-base.signedtransactions.Signed\_Transaction method*), 225
- verify\_authority()* (*bitshares.aio.transactionbuilder.TransactionBuilder method*), 85
- verify\_authority()* (*bitshares.transactionbuilder.TransactionBuilder method*), 167
- Vesting* (*class in bitshares.aio.vesting*), 86
- Vesting* (*class in bitshares.vesting*), 167
- Vesting\_balance\_withdraw* (*class in bitshares-base.operations*), 220
- vesting\_balance\_withdraw()* (*bitshares.aio.bitshares.BitShares method*), 40
- vesting\_balance\_withdraw()* (*bitshares.bitshares.BitShares method*), 119
- volume24h()* (*bitshares.aio.market.Market method*), 66
- volume24h()* (*bitshares.market.Market method*), 146
- ## W
- wait\_for\_and\_get\_block()* (*bitshares.aio.blockchain.Blockchain method*), 47
- wait\_for\_and\_get\_block()* (*bitshares.blockchain.Blockchain method*), 126
- Wallet* (*class in bitshares.aio.wallet*), 88
- Wallet* (*class in bitshares.wallet*), 169

`weight` (*bitshares.aio.witness.Witness attribute*), 91  
`weight` (*bitshares.witness.Witness attribute*), 173  
`white_and_black_listed` (*bitshares-base.operations.Account\_whitelist attribute*), 194  
`white_listed` (*bitshares-base.operations.Account\_whitelist attribute*), 194  
`whitelist()` (*bitshares.account.Account method*), 100  
`whitelist()` (*bitshares.aio.account.Account method*), 22  
`wipe()` (*bitshares.aio.wallet.Wallet method*), 89  
`wipe()` (*bitshares.wallet.Wallet method*), 171  
`with_traceback()` (*bitshares.exceptions.AccountExistsException method*), 134  
`with_traceback()` (*bitshares.exceptions.HtlcDoesNotExistException method*), 135  
`with_traceback()` (*bitshares.exceptions.ObjectNotInProposalBuffer method*), 135  
`with_traceback()` (*bitshares.exceptions.RPCConnectionRequired method*), 135  
`withdraw_from_liquidity_pool()` (*bitshares.aio.bitshares.BitShares method*), 40  
`withdraw_from_liquidity_pool()` (*bitshares.bitshares.BitShares method*), 119  
`Withdraw_permission_create` (*class in bitsharesbase.operations*), 221  
`Witness` (*class in bitshares.aio.witness*), 90  
`Witness` (*class in bitshares.witness*), 171  
`Witness_update` (*class in bitsharesbase.operations*), 222  
`Witnesses` (*class in bitshares.aio.witness*), 91  
`Witnesses` (*class in bitshares.witness*), 173  
`Worker` (*class in bitshares.aio.worker*), 93  
`Worker` (*class in bitshares.worker*), 175  
`Worker_create` (*class in bitsharesbase.operations*), 222  
`Worker_initializer` (*class in bitshares-base.objects*), 191  
`Workers` (*class in bitshares.aio.worker*), 95  
`Workers` (*class in bitshares.worker*), 177