
Privex JsonRPC Emulators Documentation

Privex Inc., Chris (Someguy123)

Oct 29, 2019

MAIN:

1	Quickstart	3
2	Example Usages	5
2.1	Using a JsonRPC emulator in a unit test	5
2.2	Using a JsonRPC emulator in your code, with a Context Manager	5
3	Python Module Overview	7
4	All Documentation	9
4.1	Installation	9
4.1.1	Download and install from PyPi using pip (recommended)	9
4.1.2	(Alternative) Manual install from Git	9
4.2	Example Usages	9
4.2.1	Using a JsonRPC emulator in a unit test	9
4.2.2	Using a JsonRPC emulator in your code, with a Context Manager	10
4.3	privex.rpcemulator.bitcoin	10
4.3.1	Attributes	14
4.3.1.1	fake	14
4.3.1.2	internal	14
4.3.2	Classes	14
4.3.2.1	BitcoinEmulator	14
4.3.2.1.1	Methods	15
4.3.2.1.1.1	__init__	15
4.3.2.1.1.2	terminate	16
4.3.3	Functions	16
4.3.3.1	getbalance	16
4.3.3.2	getblockchaininfo	17
4.3.3.3	getnetworkinfo	17
4.3.3.4	getnewaddress	17
4.3.3.5	getreceivedbyaddress	17
4.3.3.6	j_add_tx	17
4.3.3.7	j_gen_tx	18
4.3.3.8	j_transactions	18
4.3.3.9	j_update_blockchaininfo	18
4.3.3.10	j_update_networkinfo	18
4.3.3.11	listtransactions	18
4.3.3.12	sendtoaddress	19
4.4	privex.rpcemulator.base	19
4.4.1	quiet_serve	20
4.4.2	_serve	20

4.4.3	Emulator	20
4.4.3.1	Methods	20
4.4.3.1.1	__init__	21
4.4.3.1.2	terminate	21
4.4.3.2	Attributes	21
4.4.3.2.1	quiet	22
4.4.3.2.2	use_coverage	22
4.4.4	QuietRequestHandler	22
4.4.4.1	Methods	22
4.4.4.1.1	__init__	23
4.4.4.1.2	address_string	23
4.4.4.1.3	date_time_string	23
4.4.4.1.4	do_POST	23
4.4.4.1.5	end_headers	23
4.4.4.1.6	finish	23
4.4.4.1.7	flush_headers	23
4.4.4.1.8	handle	23
4.4.4.1.9	handle_expect_100	24
4.4.4.1.10	handle_one_request	24
4.4.4.1.11	log_date_time_string	24
4.4.4.1.12	log_error	24
4.4.4.1.13	log_message	24
4.4.4.1.14	log_request	24
4.4.4.1.15	parse_request	25
4.4.4.1.16	send_error	25
4.4.4.1.17	send_header	25
4.4.4.1.18	send_response	25
4.4.4.1.19	send_response_only	25
4.4.4.1.20	setup	25
4.4.4.1.21	version_string	26
4.4.4.2	Attributes	26
4.4.4.2.1	default_request_version	26
4.4.4.2.2	disable_nagle_algorithm	26
4.4.4.2.3	error_content_type	26
4.4.4.2.4	error_message_format	26
4.4.4.2.5	monthname	26
4.4.4.2.6	protocol_version	26
4.4.4.2.7	rbufsize	27
4.4.4.2.8	responses	27
4.4.4.2.9	server_version	27
4.4.4.2.10	sys_version	27
4.4.4.2.11	timeout	27
4.4.4.2.12	wbufsize	27
4.4.4.2.13	weekdayname	27
4.5	How to use the unit tests	28
4.5.1	Testing pre-requisites	28
4.5.2	Running via PyTest	28
4.5.3	Running directly using Python Unittest	28
4.6	Unit Test List / Overview	30
4.6.1	tests.test_bitcoin	30
4.6.1.1	TestBitcoinEmulator	30
4.6.1.1.1	Methods	30
4.6.1.1.1.1	setUpClass	31
4.6.1.1.1.2	tearDownClass	31

4.6.1.1.1.3	test_getblockchaininfo	31
4.6.1.1.1.4	test_getnetworkinfo	31
4.6.1.1.1.5	test_getnewaddress	31
4.6.1.1.1.6	test_send_valid	31
4.6.1.1.2	Attributes	32
4.6.1.1.2.1	EXTERNAL_ADDRESS	32
4.6.1.1.2.2	LOCAL_ADDRESS	32
4.6.1.1.2.3	rpc	32
5	Indices and tables	35
	Python Module Index	37
	Index	39

Welcome to the documentation for [Privex's JsonRPC Emulators](#) - a package designed to emulate common JsonRPC APIs, such as `bitcoind`'s JsonRPC, allowing for unit/integration testing RPC-reliant code, without needing the appropriate daemon installed (which could require a lot of configuration, synchronisation etc.).

This documentation is automatically kept up to date by ReadTheDocs, as it is automatically re-built each time a new commit is pushed to the [Github Project](#)

Contents

- *Privex JsonRPC Emulators documentation*
 - *Quickstart*
- *Example Usages*
 - *Using a JsonRPC emulator in a unit test*
 - *Using a JsonRPC emulator in your code, with a Context Manager*
- *Python Module Overview*
- *All Documentation*
- *Indices and tables*

QUICKSTART

Installing with [Pipenv](#) (recommended)

```
pipenv install rpcemulator
```

Installing with standard `pip3`

```
pip3 install rpcemulator
```


EXAMPLE USAGES

2.1 Using a JsonRPC emulator in a unit test

```
import unittest
from privex.rpcemulator.bitcoin import BitcoinEmulator
from privex.jsonrpc import BitcoinRPC

class TestMyThing(unittest.TestCase):
    emulator: BitcoinEmulator
    """Stores the :class:`.BitcoinEmulator` instance"""
    rpc = BitcoinRPC()
    """For this example, we're using our BitcoinRPC class and communicating with the
    ↪RPC directly"""

    @classmethod
    def setUpClass(cls) -> None:
        """Launch the Bitcoin RPC emulator in the background on default port 8332"""
        cls.emulator = BitcoinEmulator()

    @classmethod
    def tearDownClass(cls) -> None:
        """Shutdown the Bitcoin RPC emulator process"""
        cls.emulator.terminate()

    def test_something(self):
        """Run whatever code depends on a Bitcoin RPC"""
        self.assertGreater(self.rpc.getbalance(), 0)
```

2.2 Using a JsonRPC emulator in your code, with a Context Manager

Use the appropriate emulator class with a `with` statement so the server is automatically stopped once you're done querying it.

This prevents any risk of the web server process being leftover.

```
from privex.rpcemulator.bitcoin import BitcoinEmulator
from privex.jsonrpc import BitcoinRPC

rpc = BitcoinRPC()
print('Starting BitcoinEmulator')
```

(continues on next page)

(continued from previous page)

```
with BitcoinEmulator():  
    print('Balance is:', rpc.getbalance())  
    print('Network info is:', rpc.getnetworkinfo())  
  
print('Stopped BitcoinEmulator')
```

PYTHON MODULE OVERVIEW

Below is a listing of the sub-modules available in `rpcemulator` with a short description of what each module contains.

<code>privex.rpcemulator.bitcoin</code>	Bitcoin RPC emulator - emulates a limited bitcoind JsonRPC API
<code>privex.rpcemulator.base</code>	

ALL DOCUMENTATION

4.1 Installation

4.1.1 Download and install from PyPi using pip (recommended)

```
pipenv install rpcemulator      # Using pipenv
pip3 install rpcemulator        # Using normal pip
```

4.1.2 (Alternative) Manual install from Git

Option 1 - Use pip to install straight from Github

```
pip3 install git+https://github.com/Privex/rpcemulator
```

Option 2 - Clone and install manually

```
# Clone the repository from Github
git clone https://github.com/Privex/rpcemulator
cd rpcemulator

# RECOMMENDED MANUAL INSTALL METHOD
# Use pip to install the source code
pip3 install .

# ALTERNATIVE MANUAL INSTALL METHOD
# If you don't have pip, or have issues with installing using it, then you can use ↪
↪setuptools instead.
python3 setup.py install
```

4.2 Example Usages

4.2.1 Using a JsonRPC emulator in a unit test

```
import unittest
from privex.rpcemulator.bitcoin import BitcoinEmulator
from privex.jsonrpc import BitcoinRPC

class TestMyThing(unittest.TestCase):
```

(continues on next page)

(continued from previous page)

```

emulator: BitcoinEmulator
    """Stores the :class:`.BitcoinEmulator` instance"""
rpc = BitcoinRPC()
    """For this example, we're using our BitcoinRPC class and communicating with the
    ↪RPC directly"""

    @classmethod
    def setUpClass(cls) -> None:
        """Launch the Bitcoin RPC emulator in the background on default port 8332"""
        cls.emulator = BitcoinEmulator()

    @classmethod
    def tearDownClass(cls) -> None:
        """Shutdown the Bitcoin RPC emulator process"""
        cls.emulator.terminate()

    def test_something(self):
        """Run whatever code depends on a Bitcoin RPC"""
        self.assertGreater(self.rpc.getbalance(), 0)

```

4.2.2 Using a JsonRPC emulator in your code, with a Context Manager

Use the appropriate emulator class with a `with` statement so the server is automatically stopped once you're done querying it.

This prevents any risk of the web server process being leftover.

```

from privex.rpcemulator.bitcoin import BitcoinEmulator
from privex.jsonrpc import BitcoinRPC

rpc = BitcoinRPC()
print('Starting BitcoinEmulator')

with BitcoinEmulator():
    print('Balance is:', rpc.getbalance())
    print('Network info is:', rpc.getnetworkinfo())

print('Stopped BitcoinEmulator')

```

`privex.rpcemulator.bitcoin`

Bitcoin RPC emulator - emulates a limited bitcoind
JsonRPC API

`privex.rpcemulator.base`

4.3 privex.rpcemulator.bitcoin

Bitcoin RPC emulator - emulates a limited bitcoind JsonRPC API

While the emulation isn't complete (at the time of writing), nor does it perfectly emulate bitcoind, it's still very close, and implements methods such as `sendtoaddress()` with balance checking, address "validation", and automatically stores the send transaction (and receive TX if internal address).

To allow the RPC to be usable immediately, three receive transactions are included by default inside of `internal` - allowing you to send from these addresses with no additional configuration.

- 1PNgW6AgPZMys844kFS2dK4tt7F36MzLC8 has 0.10 BTC
- 1CGzMWXH6JhSKrkrbcGhRtEJxrU1za23LW has 0.05 BTC
- 13LWnGV7fGCUA2a9QiByGFKXL27H1HDuYp has 0.03 BTC

Basic Usage:

```
>>> from privex.rpcemulator.bitcoin import BitcoinEmulator
>>> btc_rpc = BitcoinEmulator()
>>> # make some queries to the RPC at https://127.0.0.1:8332
>>> from privex.jsonrpc import BitcoinRPC
>>> jr = BitcoinRPC()
>>> print('Balance is:', jr.getbalance())
>>> # once you're done, terminate the process
>>> btc_rpc.terminate()
```

class privex.rpcemulator.bitcoin.**BitcoinEmulator** (*host=""*, *port: int = 8332*, *background=True*)

Process manager class for the bitcoind emulator web server.

Without any constructor arguments, will fork into background at <http://127.0.0.1:8332>

By default, background is set to True, meaning it will launch as a sub-process, instead of blocking your application.

Using with a Context Manager:

By using *BitcoinEmulator* as a context manager, the JsonRPC server will be started before the first line inside of the with statement, and will automatically shutdown at the end of the with statement.

```
>>> from privex.rpcemulator.bitcoin import BitcoinEmulator
>>>
>>> with BitcoinEmulator():
...     # make some queries to the RPC at https://127.0.0.1:8332
...
>>> # Once the `with` statement is over, the JsonRPC server automatically shuts
↳down
```

Alternative

You can create an instance of *BitcoinEmulator* normally, but you should make sure to call *terminate()* when you're done with using the emulator.

This may be preferable when using inside of a unit test which has a *setUpClass* and *tearDownClass* method:

```
>>> from privex.rpcemulator.bitcoin import BitcoinEmulator
>>> btc_rpc = BitcoinEmulator()
>>> # make some queries to the RPC at https://127.0.0.1:8332
>>> # once you're done, terminate the process
>>> btc_rpc.terminate()
```

privex.rpcemulator.bitcoin.**fake** = <faker.generator.Generator object>

An instance of *faker.Faker* for generating fake data in functions such as *j_gen_tx()*

privex.rpcemulator.bitcoin.**getbalance** (*account='*'*, *confirmations: int = 0*, *watch_only=False*)

Get the balance of the RPC node, or an individual account.

Parameters

- **account** (*str*) – Only get the balance for this account. "*" or "" will sum all accounts.

- **confirmations** (*str*) – Only include transactions with at least this many confirmations
- **watch_only** – NOT IMPLEMENTED

Return float balance The total balance as a float

```
privex.rpcemulator.bitcoin.getblockchaininfo()  
Return bitcoind blockchain information, e.g. current block height
```

```
privex.rpcemulator.bitcoin.getnetworkinfo()  
Return bitcoind network information, e.g. coin daemon version
```

```
privex.rpcemulator.bitcoin.getnewaddress(account="", address_type=None)  
Generate a Bitcoin address. Note: this is simulated, it just pulls a random address from  
internal['addresses']
```

```
privex.rpcemulator.bitcoin.getreceivedbyaddress(address, confirmations: int = 0)  
Returns the total amount of coins received by address (excludes send transactions!)
```

```
privex.rpcemulator.bitcoin.internal = {'addresses': ['13LWnGV7fGCUA2a9QiByGFKXL27H1HDuYp']  
This module attribute is used as in-memory storage for various data, such as:
```

- **transactions** - A list of incoming and outgoing wallet transactions. Some are pre-defined to ensure some addresses have a balance for immediate usage of the emulator.
- **addresses** - Addresses in the emulated “wallet” that are owned by the emulated daemon
- **external_addresses** - External/foreign addresses (i.e. not controlled by this wallet). Used for very basic address validation.
- **getblockchaininfo** - Stores the dictionary that would be returned by a `getblockchaininfo()` call
- **getnetworkinfo** - Stores the dictionary that would be returned by a `getnetworkinfo()` call

```
privex.rpcemulator.bitcoin.j_add_tx(account="", address=None, amount: Union[float, str,  
decimal.Decimal] = None, category: str = None,  
**kwargs)
```

Generate a transaction using `j_gen_tx()` using the passed arguments, then store it into the transaction list.

Parameters

- **account** – Wallet account to label the transaction under
- **address** – Our address, that we’re sending from or receiving into.
- **amount** – The amount of BTC transferred
- **category** – Either 'receive' or 'send'
- **kwargs** – Any additional dict keys to put into the TX data

Return dict tx The generated TX

```
privex.rpcemulator.bitcoin.j_gen_tx(account="", address=None, amount=None, cate-  
gory=None, **kwargs)
```

Generate a Bitcoin transaction and return it as a dict.

If any transaction attributes aren’t specified, fake data will be automatically generated using `random` or `faker` to fill the attributes.

Parameters

- **account** – Wallet account to label the transaction under
- **address** – Our address, that we’re sending from or receiving into.

- **amount** – The amount of BTC transferred
- **category** – Either 'receive' or 'send'
- **kwargs** – Any additional dict keys to put into the TX data

Return dict tx The generated TX

`privex.rpcemulator.bitcoin.j_transactions (cast_decimal=<class 'float'>) → List[dict]`
Returns `internal['transactions']` with unserializable types such as `Decimal` casted appropriately.

This should be used instead of `internal['transactions']` if returning TXs from the RPC.

Parameters `cast_decimal` – A casting function to use to convert `Decimal`'s, e.g. `float` or `str`

Return List[dict] txs A list of dict transactions, with values converted to allow JSON serialisation.

`privex.rpcemulator.bitcoin.j_update_blockchaininfo (**kwargs)`
Update keys in the `blockchaininfo` using the `kwargs`

`privex.rpcemulator.bitcoin.j_update_networkinfo (**kwargs)`
Update keys in the `networkinfo` using the `kwargs`

`privex.rpcemulator.bitcoin.listtransactions (account='*', count: int = 10, skip: int = 0, watch_only=False)`
Simulates a Bitcoin RPC `listtransactions` call - returns a list of dictionary transactions from `internal['transactions']`

Parameters

- **account** – Account to list TXs for
- **count** – Load this many recent TXs
- **skip** – Skip this many recent TXs (for pagination)
- **watch_only** – (NOT IMPLEMENTED)

Returns [{account, address, category, amount, label, vout, fee, confirmations, trusted, generated, txid, time, comment, to}, ...]

`privex.rpcemulator.bitcoin.sendtoaddress (address, amount: Union[float, str, decimal.Decimal], comment="", comment_to="", subtractfee: bool = False)`

Sends amount BTC to address - generates a fake TX in internal transaction storage.

Example:

```
$ curl -v -s --data '{"method": "sendtoaddress",
  "params": ["1J2VishkhGviaEZA5dYgrqW1bjV8JGKFj", "0.001", "", "", false],
  "jsonrpc": "2.0", "id": 1}' http://127.0.0.1:5000

{"jsonrpc": "2.0", "result":
  ↪ "a4415c4013d2ba58106795ecb36a8694a3e93a4056e39ace4adde80d083c9641", "id": 1}
```

Parameters

- **address** – The destination Bitcoin address
- **amount** – The amount to send to address
- **comment** (*str*) – A comment used to store what the transaction is for.
- **comment_to** (*str*) – A comment, representing the name of the person or organization you're sending to.

- **subtractfee** (*bool*) – (Default False) If set to True, reduce the sending amount to cover the TX fee.

Returns

4.3.1 Attributes

Attributes

<i>fake</i>	An instance of <code>faker.Faker</code> for generating fake data in functions such as <code>j_gen_tx()</code>
<i>internal</i>	This module attribute is used as in-memory storage for various data, such as:

4.3.1.1 fake

```
privex.rpcemulator.bitcoin.fake = <faker.generator.Generator object>
```

An instance of `faker.Faker` for generating fake data in functions such as `j_gen_tx()`

4.3.1.2 internal

```
privex.rpcemulator.bitcoin.internal = {'addresses': ['13LWnGV7fGCUA2a9QiByGFKXL27H1HD']}
```

This module attribute is used as in-memory storage for various data, such as:

- `transactions` - A list of incoming and outgoing wallet transactions. Some are pre-defined to ensure some addresses have a balance for immediate usage of the emulator.
- `addresses` - Addresses in the emulated “wallet” that are owned by the emulated daemon
- `external_addresses` - External/foreign addresses (i.e. not controlled by this wallet). Used for very basic address validation.
- `getblockchaininfo` - Stores the dictionary that would be returned by a `getblockchaininfo()` call
- `getnetworkinfo` - Stores the dictionary that would be returned by a `getnetworkinfo()` call

4.3.2 Classes

Classes

<i>BitcoinEmulator</i> ([host, port, background])	Process manager class for the bitcoind emulator web server.
---	---

4.3.2.1 BitcoinEmulator

```
class privex.rpcemulator.bitcoin.BitcoinEmulator (host="", port: int = 8332, background=True)
```

Process manager class for the bitcoind emulator web server.

Without any constructor arguments, will fork into background at <http://127.0.0.1:8332>

By default, `background` is set to `True`, meaning it will launch as a sub-process, instead of blocking your application.

Using with a Context Manager:

By using `BitcoinEmulator` as a context manager, the JsonRPC server will be started before the first line inside of the `with` statement, and will automatically shutdown at the end of the `with` statement.

```
>>> from privex.rpcemulator.bitcoin import BitcoinEmulator
>>>
>>> with BitcoinEmulator():
...     # make some queries to the RPC at https://127.0.0.1:8332
...
>>> # Once the `with` statement is over, the JsonRPC server_
↳ automatically shuts down
```

Alternative

You can create an instance of `BitcoinEmulator` normally, but you should make sure to call `terminate()` when you're done with using the emulator.

This may be preferable when using inside of a unit test which has a `setUpClass` and `tearDownClass` method:

```
>>> from privex.rpcemulator.bitcoin import BitcoinEmulator
>>> btc_rpc = BitcoinEmulator()
>>> # make some queries to the RPC at https://127.0.0.1:8332
>>> # once you're done, terminate the process
>>> btc_rpc.terminate()
```

4.3.2.1.1 Methods

Methods

<code>__init__([host, port, background])</code>	Without any constructor arguments, will fork into background at http://127.0.0.1:8332
<code>terminate()</code>	Called when a user wants to manually terminate the background process.

4.3.2.1.1.1 `__init__`

`BitcoinEmulator.__init__(host="", port: int = 8332, background=True)`

Without any constructor arguments, will fork into background at <http://127.0.0.1:8332>

By default, `background` is set to `True`, meaning it will launch as a sub-process, instead of blocking your application.

```
>>> from privex.rpcemulator.bitcoin import BitcoinEmulator
>>>
>>> with BitcoinEmulator():
...     # make some queries to the RPC at https://127.0.0.1:8332
...
>>> # Once the `with` statement is over, the JsonRPC server_
↳ automatically shuts down
```

Parameters

- **host** (*str*) – The IP address to listen on. If left as "" - will listen at 127.0.0.1
- **port** (*int*) – The port number to listen on (Defaults to 8332, same as Bitcoin)
- **background** (*bool*) – If True, spawns the webserver in a sub-process, instead of blocking the app.

4.3.2.1.1.2 terminate

BitcoinEmulator.**terminate**()

Called when a user wants to manually terminate the background process.

Simply calls `__del__()` to terminate the process.

4.3.3 Functions

Functions

<code>getbalance([account, confirmations, watch_only])</code>	Get the balance of the RPC node, or an individual account.
<code>getblockchaininfo()</code>	Return bitcoind blockchain information, e.g.
<code>getnetworkinfo()</code>	Return bitcoind network information, e.g.
<code>getnewaddress([account, address_type])</code>	Generate a Bitcoin address.
<code>getreceivedbyaddress(address[, confirmations])</code>	Returns the total amount of coins received by address (excludes send transactions!)
<code>j_add_tx([account, address, amount, category])</code>	Generate a transaction using <code>j_gen_tx()</code> using the passed arguments, then store it into the transaction list.
<code>j_gen_tx([account, address, amount, category])</code>	Generate a Bitcoin transaction and return it as a dict.
<code>j_transactions([cast_decimal])</code>	Returns <code>internal['transactions']</code> with unserializable types such as Decimal casted appropriately.
<code>j_update_blockchaininfo(**kwargs)</code>	Update keys in the blockchaininfo using the kwargs
<code>j_update_networkinfo(**kwargs)</code>	Update keys in the networkinfo using the kwargs
<code>listtransactions([account, count, skip, ...])</code>	Simulates a Bitcoin RPC <code>listtransactions</code> call - returns a list of dictionary transactions from <code>internal['transactions']</code>
<code>sendtoaddress(address, amount[, comment, ...])</code>	Sends amount BTC to address - generates a fake TX in internal transaction storage.

4.3.3.1 getbalance

`privex.rpcemulator.bitcoin.getbalance(account='*', confirmations: int = 0, watch_only=False)`

Get the balance of the RPC node, or an individual account.

Parameters

- **account** (*str*) – Only get the balance for this account. "*" or "" will sum all accounts.

- **confirmations** (*str*) – Only include transactions with at least this many confirmations
- **watch_only** – NOT IMPLEMENTED

Return float balance The total balance as a float

4.3.3.2 getblockchaininfo

```
privex.rpcemulator.bitcoin.getblockchaininfo()
```

Return bitcoind blockchain information, e.g. current block height

4.3.3.3 getnetworkinfo

```
privex.rpcemulator.bitcoin.getnetworkinfo()
```

Return bitcoind network information, e.g. coin daemon version

4.3.3.4 getnewaddress

```
privex.rpcemulator.bitcoin.getnewaddress(account="", address_type=None)
```

Generate a Bitcoin address. Note: this is simulated, it just pulls a random address from `internal['addresses']`

4.3.3.5 getreceivedbyaddress

```
privex.rpcemulator.bitcoin.getreceivedbyaddress(address, confirmations: int = 0)
```

Returns the total amount of coins received by `address` (excludes send transactions!)

4.3.3.6 j_add_tx

```
privex.rpcemulator.bitcoin.j_add_tx(account="", address=None, amount: Union[float, str, decimal.Decimal] = None, category: str = None, **kwargs)
```

Generate a transaction using `j_gen_tx()` using the passed arguments, then store it into the transaction list.

Parameters

- **account** – Wallet account to label the transaction under
- **address** – Our address, that we're sending from or receiving into.
- **amount** – The amount of BTC transferred
- **category** – Either 'receive' or 'send'
- **kwargs** – Any additional dict keys to put into the TX data

Return dict tx The generated TX

4.3.3.7 j_gen_tx

```
privex.rpcemulator.bitcoin.j_gen_tx(account="", address=None, amount=None,  
                                     category=None, **kwargs)
```

Generate a Bitcoin transaction and return it as a dict.

If any transaction attributes aren't specified, fake data will be automatically generated using `random` or `faker` to fill the attributes.

Parameters

- **account** – Wallet account to label the transaction under
- **address** – Our address, that we're sending from or receiving into.
- **amount** – The amount of BTC transferred
- **category** – Either 'receive' or 'send'
- **kwargs** – Any additional dict keys to put into the TX data

Return dict tx The generated TX

4.3.3.8 j_transactions

```
privex.rpcemulator.bitcoin.j_transactions(cast_decimal=<class 'float'>) →  
                                         List[dict]
```

Returns `internal['transactions']` with unserializable types such as `Decimal` casted appropriately.

This should be used instead of `internal['transactions']` if returning TXs from the RPC.

Parameters cast_decimal – A casting function to use to convert `Decimal`'s, e.g. `float` or `str`

Return List[dict] txs A list of dict transactions, with values converted to allow JSON serialisation.

4.3.3.9 j_update_blockchaininfo

```
privex.rpcemulator.bitcoin.j_update_blockchaininfo(**kwargs)
```

Update keys in the `blockchaininfo` using the `kwargs`

4.3.3.10 j_update_networkinfo

```
privex.rpcemulator.bitcoin.j_update_networkinfo(**kwargs)
```

Update keys in the `networkinfo` using the `kwargs`

4.3.3.11 listtransactions

```
privex.rpcemulator.bitcoin.listtransactions(account='', count: int = 10,  
                                             skip: int = 0, watch_only=False)
```

Simulates a Bitcoin RPC `listtransactions` call - returns a list of dictionary transactions from `internal['transactions']`

Parameters

- **account** – Account to list TXs for

- **count** – Load this many recent TXs
- **skip** – Skip this many recent TXs (for pagination)
- **watch_only** – (NOT IMPLEMENTED)

Returns [{account, address, category, amount, label, vout, fee, confirmations, trusted, generated, txid, time, comment, to}, ...]

4.3.3.12 sendtoaddress

`privex.rpcemulator.bitcoin.sendtoaddress` (*address*, *amount*: *Union[float, str, decimal.Decimal]*, *comment*=", *comment_to*", *subtractfee*: *bool* = *False*)

Sends amount BTC to address - generates a fake TX in internal transaction storage.

Example:

```
$ curl -v -s --data '{"method": "sendtoaddress",
  "params": ["1J2VishkhGviaEZA5dYgrqW1bjV8JGKFj", "0.001", "", "",
  ↪false],
  "jsonrpc": "2.0", "id": 1}' http://127.0.0.1:5000

{"jsonrpc": "2.0", "result":
  ↪"a4415c4013d2ba58106795ecb36a8694a3e93a4056e39ace4adde80d083c9641", "id
  ↪": 1}
```

Parameters

- **address** – The destination Bitcoin address
- **amount** – The amount to send to address
- **comment** (*str*) – A comment used to store what the transaction is for.
- **comment_to** (*str*) – A comment, representing the name of the person or organization you're sending to.
- **subtractfee** (*bool*) – (Default False) If set to True, reduce the sending amount to cover the TX fee.

Returns

4.4 privex.rpcemulator.base

Functions

<code>quiet_serve([name, port])</code>	Quiet version of <code>jsonrpcserver.serve()</code> with logging disabled.
<code>_serve([host, port, quiet, use_coverage])</code>	Wrapper function for <code>jsonrpcserver.serve()</code> and <code>quiet_serve()</code> .

4.4.1 quiet_serve

`privex.rpcemulator.base.quiet_serve(name: str = "", port: int = 5000) → None`
Quiet version of `jsonrpcserver.serve()` with logging disabled.

Args: name: Server address. port: Server port.

4.4.2 _serve

`privex.rpcemulator.base._serve(host="", port=5000, quiet=False, use_coverage=False)`
Wrapper function for `jsonrpcserver.serve()` and `quiet_serve()`. Can be forked into background.
Sets up SIGTERM hook using `pytest_cov.embed.cleanup_on_sigterm()` so coverage data is correctly saved when the subprocess is terminated.

Classes

<code>Emulator([host, port, background])</code>	This is the base class used by JsonRPC emulators such as <code>privex.rpcemulator.bitcoin.BitcoinEmulator</code>
<code>QuietRequestHandler(request, client_address, ...)</code>	Same as <code>jsonrpcserver.server.RequestHandler</code> but with logging disabled.

4.4.3 Emulator

class `privex.rpcemulator.base.Emulator(host="", port: int = 5000, background=True)`
This is the base class used by JsonRPC emulators such as `privex.rpcemulator.bitcoin.BitcoinEmulator`

It fires `jsonrpcserver.serve()` into the background using `multiprocessing` and handles shutting down the process either via context management (with statements), direct calls to `terminate()`, or when the object is garbage collected via `__del__()`

proc = None

Holds the `multiprocessing.Process` background process instance for `serve()`

quiet = True

Set `Emulator.quiet = True` to use `quiet_serve()` (disable HTTP request logging)

terminate()

Called when a user wants to manually terminate the background process.

Simply calls `__del__()` to terminate the process.

use_coverage = True

When running unit tests, this should be set to `True` to load coverage in the subprocess

4.4.3.1 Methods

Methods

<code>__init__([host, port, background])</code>	Launch an RPC emulator web server.
---	------------------------------------

Continued on next page

Table 8 – continued from previous page

<code>terminate()</code>	Called when a user wants to manually terminate the background process.
--------------------------	--

4.4.3.1.1 `__init__`

`Emulator.__init__(host="", port: int = 5000, background=True)`

Launch an RPC emulator web server. Without arguments, will fork into background at <http://127.0.0.1:5000>

By default, `background` is set to `True`, meaning it will launch as a sub-process, instead of blocking your application. You can use the returned `multiprocessing.Process` object to terminate it once you're done using it.

Using with a Context Manager::

```
>>> from privex.rpcemulator.base import Emulator
>>>
>>> with Emulator():
...     # make some queries to the RPC at https://127.0.0.1:5000
...
>>> # Once the `with` statement is over, the JsonRPC server automatically_
↳ shuts down
```

Example:

```
>>> from privex.rpcemulator.base import Emulator
>>> rpc = Emulator()
>>> # make some queries to the RPC at https://127.0.0.1:5000
>>> # once you're done, terminate the process
>>> rpc.terminate()
```

Parameters

- **host** (*str*) – The IP address to listen on. If left as "" - will listen at 127.0.0.1
- **port** (*int*) – The port number to listen on (Defaults to 5000)
- **background** (*bool*) – If `True`, spawns the webserver in a sub-process, instead of blocking the app.

4.4.3.1.2 `terminate`

`Emulator.terminate()`

Called when a user wants to manually terminate the background process.

Simply calls `__del__()` to terminate the process.

4.4.3.2 Attributes

Attributes

<code>quiet</code>	Set <code>Emulator.quiet = True</code> to use <code>quiet_serve()</code> (disable HTTP request logging)
--------------------	---

Continued on next page

Table 9 – continued from previous page

<code>use_coverage</code>	When running unit tests, this should be set to True to load coverage in the subprocess
---------------------------	--

4.4.3.2.1 quiet

`Emulator.quiet = True`

Set `Emulator.quiet = True` to use `quiet_serve()` (disable HTTP request logging)

4.4.3.2.2 use_coverage

`Emulator.use_coverage = True`

When running unit tests, this should be set to True to load coverage in the subprocess

4.4.4 QuietRequestHandler

class `privex.rpcemulator.base.QuietRequestHandler` (*request, client_address, server*)

Same as `jsonrpcserver.server.RequestHandler` but with logging disabled.

log_message (*format, *args*)

Log an arbitrary message.

This is used by all other logging functions. Override it if you have specific logging wishes.

The first argument, `FORMAT`, is a format string for the message to be logged. If the format string contains any `%` escapes requiring parameters, they should be specified as subsequent arguments (it's just like `printf!`).

The client ip and current date/time are prefixed to every message.

4.4.4.1 Methods

Methods

<code>__init__(request, client_address, server)</code>	Initialize self.
<code>address_string()</code>	Return the client address.
<code>date_time_string([timestamp])</code>	Return the current date and time formatted for a message header.
<code>do_POST()</code>	HTTP POST
<code>end_headers()</code>	Send the blank line ending the MIME headers.
<code>finish()</code>	
<code>flush_headers()</code>	
<code>handle()</code>	Handle multiple requests if necessary.
<code>handle_expect_100()</code>	Decide what to do with an "Expect: 100-continue" header.
<code>handle_one_request()</code>	Handle a single HTTP request.
<code>log_date_time_string()</code>	Return the current time formatted for logging.
<code>log_error(format, *args)</code>	Log an error.
<code>log_message(format, *args)</code>	Log an arbitrary message.
<code>log_request([code, size])</code>	Log an accepted request.
<code>parse_request()</code>	Parse a request (internal).

Continued on next page

Table 10 – continued from previous page

<code>send_error</code> (code[, message, explain])	Send and log an error reply.
<code>send_header</code> (keyword, value)	Send a MIME header to the headers buffer.
<code>send_response</code> (code[, message])	Add the response header to the headers buffer and log the response code.
<code>send_response_only</code> (code[, message])	Send the response header only.
<code>setup</code> ()	
<code>version_string</code> ()	Return the server software version string.

4.4.4.1.1 `__init__`

`QuietRequestHandler.__init__(request, client_address, server)`

Initialize self. See `help(type(self))` for accurate signature.

4.4.4.1.2 `address_string`

`QuietRequestHandler.address_string()`

Return the client address.

4.4.4.1.3 `date_time_string`

`QuietRequestHandler.date_time_string(timestamp=None)`

Return the current date and time formatted for a message header.

4.4.4.1.4 `do_POST`

`QuietRequestHandler.do_POST()` → None

HTTP POST

4.4.4.1.5 `end_headers`

`QuietRequestHandler.end_headers()`

Send the blank line ending the MIME headers.

4.4.4.1.6 `finish`

`QuietRequestHandler.finish()`

4.4.4.1.7 `flush_headers`

`QuietRequestHandler.flush_headers()`

4.4.4.1.8 `handle`

`QuietRequestHandler.handle()`

Handle multiple requests if necessary.

4.4.4.1.9 `handle_expect_100`

`QuietRequestHandler.handle_expect_100()`

Decide what to do with an “Expect: 100-continue” header.

If the client is expecting a 100 Continue response, we must respond with either a 100 Continue or a final response before waiting for the request body. The default is to always respond with a 100 Continue. You can behave differently (for example, reject unauthorized requests) by overriding this method.

This method should either return `True` (possibly after sending a 100 Continue response) or send an error response and return `False`.

4.4.4.1.10 `handle_one_request`

`QuietRequestHandler.handle_one_request()`

Handle a single HTTP request.

You normally don’t need to override this method; see the class `__doc__` string for information on how to handle specific HTTP commands such as GET and POST.

4.4.4.1.11 `log_date_time_string`

`QuietRequestHandler.log_date_time_string()`

Return the current time formatted for logging.

4.4.4.1.12 `log_error`

`QuietRequestHandler.log_error(format, *args)`

Log an error.

This is called when a request cannot be fulfilled. By default it passes the message on to `log_message()`.

Arguments are the same as for `log_message()`.

XXX This should go to the separate error log.

4.4.4.1.13 `log_message`

`QuietRequestHandler.log_message(format, *args)`

Log an arbitrary message.

This is used by all other logging functions. Override it if you have specific logging wishes.

The first argument, `FORMAT`, is a format string for the message to be logged. If the format string contains any `%` escapes requiring parameters, they should be specified as subsequent arguments (it’s just like `printf!`).

The client ip and current date/time are prefixed to every message.

4.4.4.1.14 `log_request`

`QuietRequestHandler.log_request(code='-', size='-')`

Log an accepted request.

This is called by `send_response()`.

4.4.4.1.15 parse_request

`QuietRequestHandler.parse_request()`

Parse a request (internal).

The request should be stored in `self.raw_requestline`; the results are in `self.command`, `self.path`, `self.request_version` and `self.headers`.

Return True for success, False for failure; on failure, any relevant error response has already been sent back.

4.4.4.1.16 send_error

`QuietRequestHandler.send_error(code, message=None, explain=None)`

Send and log an error reply.

Arguments are * `code`: an HTTP error code

3 digits

- **message: a simple optional 1 line reason phrase.** *(HTAB / SP / VCHAR / %x80-FF) defaults to short entry matching the response code
- **explain: a detailed message defaults to the long entry** matching the response code.

This sends an error response (so it must be called before any output has been generated), logs the error, and finally sends a piece of HTML explaining the error to the user.

4.4.4.1.17 send_header

`QuietRequestHandler.send_header(keyword, value)`

Send a MIME header to the headers buffer.

4.4.4.1.18 send_response

`QuietRequestHandler.send_response(code, message=None)`

Add the response header to the headers buffer and log the response code.

Also send two standard headers with the server software version and the current date.

4.4.4.1.19 send_response_only

`QuietRequestHandler.send_response_only(code, message=None)`

Send the response header only.

4.4.4.1.20 setup

`QuietRequestHandler.setup()`

4.4.4.1.21 version_string

`QuietRequestHandler.version_string()`
Return the server software version string.

4.4.4.2 Attributes

Attributes

<i>default_request_version</i>
<i>disable_nagle_algorithm</i>
<i>error_content_type</i>
<i>error_message_format</i>
<i>monthname</i>
<i>protocol_version</i>
<i>rbufsize</i>
<i>responses</i>
<i>server_version</i>
<i>sys_version</i>
<i>timeout</i>
<i>wbufsize</i>
<i>weekdayname</i>

4.4.4.2.1 default_request_version

`QuietRequestHandler.default_request_version = 'HTTP/0.9'`

4.4.4.2.2 disable_nagle_algorithm

`QuietRequestHandler.disable_nagle_algorithm = False`

4.4.4.2.3 error_content_type

`QuietRequestHandler.error_content_type = 'text/html; charset=utf-8'`

4.4.4.2.4 error_message_format

`QuietRequestHandler.error_message_format = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN'>'`

4.4.4.2.5 monthname

`QuietRequestHandler.monthname = [None, 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']`

4.4.4.2.6 protocol_version

`QuietRequestHandler.protocol_version = 'HTTP/1.0'`

4.4.4.2.7 rbufsize

```
QuietRequestHandler.rbufsize = -1
```

4.4.4.2.8 responses

```
QuietRequestHandler.responses = {<HTTPStatus.CONTINUE: 100>: ('Continue', 'Request received')}
```

4.4.4.2.9 server_version

```
QuietRequestHandler.server_version = 'BaseHTTP/0.6'
```

4.4.4.2.10 sys_version

```
QuietRequestHandler.sys_version = 'Python/3.7.3'
```

4.4.4.2.11 timeout

```
QuietRequestHandler.timeout = None
```

4.4.4.2.12 wbufsize

```
QuietRequestHandler.wbufsize = 0
```

4.4.4.2.13 weekdayname

```
QuietRequestHandler.weekdayname = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
```

```
class privex.rpcemulator.base.Emulator(host="", port: int = 5000, background=True)
```

This is the base class used by JsonRPC emulators such as `privex.rpcemulator.bitcoin.BitcoinEmulator`

It fires `jsonrpcserver.serve()` into the background using `multiprocessing` and handles shutting down the process either via context management (with statements), direct calls to `terminate()`, or when the object is garbage collected via `__del__()`

proc = None

Holds the `multiprocessing.Process` background process instance for `serve()`

quiet = True

Set `Emulator.quiet = True` to use `quiet_serve()` (disable HTTP request logging)

terminate()

Called when a user wants to manually terminate the background process.

Simply calls `__del__()` to terminate the process.

use_coverage = True

When running unit tests, this should be set to `True` to load coverage in the subprocess

```
class privex.rpcemulator.base.QuietRequestHandler(request, client_address, server)
```

Same as `jsonrpcserver.server.RequestHandler` but with logging disabled.

log_message (*format, *args*)

Log an arbitrary message.

This is used by all other logging functions. Override it if you have specific logging wishes.

The first argument, FORMAT, is a format string for the message to be logged. If the format string contains any % escapes requiring parameters, they should be specified as subsequent arguments (it's just like printf!).

The client ip and current date/time are prefixed to every message.

`privex.rpcemulator.base.quiet_serve` (*name: str = "", port: int = 5000*) → None

Quiet version of `jsonrpcserver.serve()` with logging disabled.

Args: name: Server address. port: Server port.

4.5 How to use the unit tests

This module contains test cases for Privex's JsonRPC Emulators (`rpcemulator`).

4.5.1 Testing pre-requisites

- Ensure you have any mandatory requirements installed (see `setup.py`'s `install_requires`)
- You may wish to install any optional requirements listed in `README.md` for best results
- Python 3.7 is recommended at the time of writing this. See `README.md` in-case this has changed.

4.5.2 Running via PyTest

To run the tests, we strongly recommend using the `pytest` tool (used by default for our Travis CI):

```
# Install requirements.txt which should include PyTest
user@host: ~/rpcemulator $ pip3 install -r requirements.txt
# You can add '-v' for more detailed output, just like when running the tests
↳ directly.
user@host: ~/rpcemulator $ pytest

===== test session starts
↳ =====
platform darwin -- Python 3.7.0, pytest-5.0.1, py-1.8.0, pluggy-0.12.0
rootdir: /home/user/rpcemulator
collected 4 items

tests/test_bitcoin.py ....
↳ [100%]

===== 4 passed, 1 warnings in 0.17 seconds
↳ =====
```

4.5.3 Running directly using Python Unittest

Alternatively, you can run the tests by hand with `python3.7` (or just `python3`)

```

user@the-matrix ~/rpceimulator $ python3.7 -m tests
....
-----
Ran 4 tests in 0.001s

OK

```

For more verbosity, simply add `-v` to the end of the command:

```

user@the-matrix ~/rpceimulator $ python3 -m tests -v
test_getblockchaininfo (tests.test_bitcoin.TestBitcoinEmulator)
Test that the ``getblockchaininfo`` JsonRPC call returns data as expected ... ok
test_getnetworkinfo (tests.test_bitcoin.TestBitcoinEmulator)
Test that the ``getnetworkinfo`` JsonRPC call returns data as expected ... ok
test_getnewaddress (tests.test_bitcoin.TestBitcoinEmulator)
Get a new address from the emulator and confirm it seems like a BTC address ... ok
test_send_valid (tests.test_bitcoin.TestBitcoinEmulator)
Test sending coins to external address creates a TX in listtransactions, and reduces
↳the balance ... ok

-----
Ran 4 tests in 0.242s

OK

```

Copyright:

```

Copyright 2019      Privex Inc.   ( https://www.privex.io )
License: X11 / MIT  Github: https://github.com/Privex/rpceimulator

```

```

+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io              |
+=====+
|                Originally Developed by Privex Inc.  |
|                Core Developer(s):                 |
|                (+) Chris (@someguy123) [Privex]    |
|                (+) Kale (@kryogenic) [Privex]     |
+=====+

```

```

Copyright 2019      Privex Inc.   ( https://www.privex.io )

```

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

(continues on next page)

(continued from previous page)

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

4.6 Unit Test List / Overview

`tests.test_bitcoin`

4.6.1 tests.test_bitcoin

Classes

`TestBitcoinEmulator([methodName])`

4.6.1.1 TestBitcoinEmulator

class tests.test_bitcoin.TestBitcoinEmulator (methodName='runTest')

EXTERNAL_ADDRESS = '13J8HRihYqEDYHAXLciryQYTjpxXcjYMmR'

A Bitcoin address considered 'foreign' for testing that sending reduces balance

LOCAL_ADDRESS = '1PNgW6AgPZMys844kFS2dK4tt7F36MzLC8'

A Bitcoin address considered to be in the wallet

emulator = None

Stores the Process returned from bitcoin.j_server()

classmethod setUpClass() → None

Launch the Bitcoin RPC emulator in the background on default port 8332

classmethod tearDownClass() → None

Shutdown the Bitcoin RPC emulator process

test_getblockchaininfo()

Test that the getblockchaininfo JsonRPC call returns data as expected

test_getnetworkinfo()

Test that the getnetworkinfo JsonRPC call returns data as expected

test_getnewaddress()

Get a new address from the emulator and confirm it seems like a BTC address

test_send_valid()

Test sending coins to external address creates a TX in listtransactions, and reduces the balance

4.6.1.1.1 Methods

Methods

<code>setUpClass()</code>	Launch the Bitcoin RPC emulator in the background on default port 8332
<code>tearDownClass()</code>	Shutdown the Bitcoin RPC emulator process
<code>test_getblockchaininfo()</code>	Test that the <code>getblockchaininfo</code> JsonRPC call returns data as expected
<code>test_getnetworkinfo()</code>	Test that the <code>getnetworkinfo</code> JsonRPC call returns data as expected
<code>test_getnewaddress()</code>	Get a new address from the emulator and confirm it seems like a BTC address
<code>test_send_valid()</code>	Test sending coins to external address creates a TX in <code>listtransactions</code> , and reduces the balance

4.6.1.1.1 setUpClass

classmethod `TestBitcoinEmulator.setUpClass()` → None
Launch the Bitcoin RPC emulator in the background on default port 8332

4.6.1.1.2 tearDownClass

classmethod `TestBitcoinEmulator.tearDownClass()` → None
Shutdown the Bitcoin RPC emulator process

4.6.1.1.3 test_getblockchaininfo

`TestBitcoinEmulator.test_getblockchaininfo()`
Test that the `getblockchaininfo` JsonRPC call returns data as expected

4.6.1.1.4 test_getnetworkinfo

`TestBitcoinEmulator.test_getnetworkinfo()`
Test that the `getnetworkinfo` JsonRPC call returns data as expected

4.6.1.1.5 test_getnewaddress

`TestBitcoinEmulator.test_getnewaddress()`
Get a new address from the emulator and confirm it seems like a BTC address

4.6.1.1.6 test_send_valid

`TestBitcoinEmulator.test_send_valid()`
Test sending coins to external address creates a TX in `listtransactions`, and reduces the balance

4.6.1.1.2 Attributes

Attributes

<code>EXTERNAL_ADDRESS</code>	A Bitcoin address considered ‘foreign’ for testing that sending reduces balance
<code>LOCAL_ADDRESS</code>	A Bitcoin address considered to be in the wallet
<code>rpc</code>	Wrapper class for JsonRPC, with default host 127.0.0.1 and port 8332 Contains pre-defined methods with pydoc for interacting with <i>bitcoind</i> compatible JsonRPC services including most coin daemons forked from Bitcoin, e.g.

4.6.1.1.2.1 EXTERNAL_ADDRESS

`TestBitcoinEmulator.EXTERNAL_ADDRESS = '13J8HRihYqEDYHAXLciryQYTjpxXcjYMmR'`
A Bitcoin address considered ‘foreign’ for testing that sending reduces balance

4.6.1.1.2.2 LOCAL_ADDRESS

`TestBitcoinEmulator.LOCAL_ADDRESS = '1PNgW6AgPZMys844kFS2dK4tt7F36MzLC8'`
A Bitcoin address considered to be in the wallet

4.6.1.1.2.3 rpc

`TestBitcoinEmulator.rpc`

Wrapper class for JsonRPC, with default host 127.0.0.1 and port 8332 Contains pre-defined methods with pydoc for interacting with *bitcoind* compatible JsonRPC services including most coin daemons forked from Bitcoin, e.g. litecoind, dogecoind etc.

If a method is not defined, you can still use it! You just won’t get any IDE hints with the parameters.

Basic usage (by default, connects to <http://127.0.0.1:8332>):

```
>>> j = BitcoinRPC(username='bitcoinrpc', password='somesecurepassword')
>>> j.getbalance()
Decimal(0.2456337)
```

`class tests.test_bitcoin.TestBitcoinEmulator (methodName='runTest')`

`EXTERNAL_ADDRESS = '13J8HRihYqEDYHAXLciryQYTjpxXcjYMmR'`
A Bitcoin address considered ‘foreign’ for testing that sending reduces balance

`LOCAL_ADDRESS = '1PNgW6AgPZMys844kFS2dK4tt7F36MzLC8'`
A Bitcoin address considered to be in the wallet

`emulator = None`
Stores the `Process` returned from `bitcoin.j_server()`

`classmethod setUpClass() → None`
Launch the Bitcoin RPC emulator in the background on default port 8332

classmethod `tearDownClass()` → None
Shutdown the Bitcoin RPC emulator process

test_`getblockchaininfo()`
Test that the `getblockchaininfo` JsonRPC call returns data as expected

test_`getnetworkinfo()`
Test that the `getnetworkinfo` JsonRPC call returns data as expected

test_`getnewaddress()`
Get a new address from the emulator and confirm it seems like a BTC address

test_`send_valid()`
Test sending coins to external address creates a TX in `listtransactions`, and reduces the balance

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

p

`privex.rpcemulator.base`, [19](#)

`privex.rpcemulator.bitcoin`, [10](#)

t

`tests`, [28](#)

`tests.test_bitcoin`, [30](#)

INDEX

Symbols

`__init__()` (*privex.rpcemulator.base.Emulator*
method), 21

`__init__()` (*privex.rpcemulator.base.QuietRequestHandler*
method), 23

`__init__()` (*privex.rpcemulator.bitcoin.BitcoinEmulator*
method), 15

`_serve()` (*in module privex.rpcemulator.base*), 20

A

`address_string()` (*privex.rpcemulator.base.QuietRequestHandler*
method), 23

B

`BitcoinEmulator` (class *in*
privex.rpcemulator.bitcoin), 11, 14

D

`date_time_string()`
(*privex.rpcemulator.base.QuietRequestHandler*
method), 23

`default_request_version`
(*privex.rpcemulator.base.QuietRequestHandler*
attribute), 26

`disable_nagle_algorithm`
(*privex.rpcemulator.base.QuietRequestHandler*
attribute), 26

`do_POST()` (*privex.rpcemulator.base.QuietRequestHandler*
method), 23

E

`Emulator` (class *in privex.rpcemulator.base*), 20, 27

`emulator` (*tests.test_bitcoin.TestBitcoinEmulator* at-
tribute), 30, 32

`end_headers()` (*privex.rpcemulator.base.QuietRequestHandler*
method), 23

`error_content_type`
(*privex.rpcemulator.base.QuietRequestHandler*
attribute), 26

`error_message_format`
(*privex.rpcemulator.base.QuietRequestHandler*
attribute), 26

`EXTERNAL_ADDRESS` (*tests.test_bitcoin.TestBitcoinEmulator*
attribute), 30, 32

F

`fake` (*in module privex.rpcemulator.bitcoin*), 11, 14

`finish()` (*privex.rpcemulator.base.QuietRequestHandler*
method), 23

`flush_headers()` (*privex.rpcemulator.base.QuietRequestHandler*
method), 23

G

`getbalance()` (*in* module
privex.rpcemulator.bitcoin), 11, 16

`getblockchaininfo()` (*in* module
privex.rpcemulator.bitcoin), 12, 17

`getnetworkinfo()` (*in* module
privex.rpcemulator.bitcoin), 12, 17

`getnewaddress()` (*in* module
privex.rpcemulator.bitcoin), 12, 17

`getreceivedbyaddress()` (*in* module
privex.rpcemulator.bitcoin), 12, 17

H

`handle()` (*privex.rpcemulator.base.QuietRequestHandler*
method), 23

`handle_expect_100()`
(*privex.rpcemulator.base.QuietRequestHandler*
method), 24

`handle_one_request()`
(*privex.rpcemulator.base.QuietRequestHandler*
method), 24

I

`internal` (*in module privex.rpcemulator.bitcoin*), 12,

J

`j_add_tx()` (*in module privex.rpcemulator.bitcoin*),
12, 17

`j_gen_tx()` (*in module privex.rpcemulator.bitcoin*),
12, 18

[j_transactions\(\)](#) (in module [privex.rpcemulator.bitcoin](#)), 13, 18
[j_update_blockchaininfo\(\)](#) (in module [privex.rpcemulator.bitcoin](#)), 13, 18
[j_update_networkinfo\(\)](#) (in module [privex.rpcemulator.bitcoin](#)), 13, 18

L

[listtransactions\(\)](#) (in module [privex.rpcemulator.bitcoin](#)), 13, 18
[LOCAL_ADDRESS](#) ([tests.test_bitcoin.TestBitcoinEmulator](#) attribute), 30, 32
[log_date_time_string\(\)](#) ([privex.rpcemulator.base.QuietRequestHandler](#) method), 24
[log_error\(\)](#) ([privex.rpcemulator.base.QuietRequestHandler](#) method), 24
[log_message\(\)](#) ([privex.rpcemulator.base.QuietRequestHandler](#) method), 22, 24, 27
[log_request\(\)](#) ([privex.rpcemulator.base.QuietRequestHandler](#) method), 24

M

[monthname](#) ([privex.rpcemulator.base.QuietRequestHandler](#) attribute), 26

P

[parse_request\(\)](#) ([privex.rpcemulator.base.QuietRequestHandler](#) method), 25
[privex.rpcemulator.base](#) (module), 19
[privex.rpcemulator.bitcoin](#) (module), 10
[proc](#) ([privex.rpcemulator.base.Emulator](#) attribute), 20, 27
[protocol_version](#) ([privex.rpcemulator.base.QuietRequestHandler](#) attribute), 26

Q

[quiet](#) ([privex.rpcemulator.base.Emulator](#) attribute), 20, 22, 27
[quiet_serve\(\)](#) (in module [privex.rpcemulator.base](#)), 20, 28
[QuietRequestHandler](#) (class in [privex.rpcemulator.base](#)), 22, 27

R

[rbufsize](#) ([privex.rpcemulator.base.QuietRequestHandler](#) attribute), 27
[responses](#) ([privex.rpcemulator.base.QuietRequestHandler](#) attribute), 27
[rpc](#) ([tests.test_bitcoin.TestBitcoinEmulator](#) attribute), 32

S

[send_error\(\)](#) ([privex.rpcemulator.base.QuietRequestHandler](#) method), 25

[send_header\(\)](#) ([privex.rpcemulator.base.QuietRequestHandler](#) method), 25
[send_response\(\)](#) ([privex.rpcemulator.base.QuietRequestHandler](#) method), 25
[send_response_only\(\)](#) ([privex.rpcemulator.base.QuietRequestHandler](#) method), 25
[sendtoaddress\(\)](#) (in module [privex.rpcemulator.bitcoin](#)), 13, 19
[server_version](#) ([privex.rpcemulator.base.QuietRequestHandler](#) attribute), 27
[setup\(\)](#) ([privex.rpcemulator.base.QuietRequestHandler](#) method), 25
[setUpClass\(\)](#) ([tests.test_bitcoin.TestBitcoinEmulator](#) class method), 30–32
[tearDownClass\(\)](#) ([tests.test_bitcoin.TestBitcoinEmulator](#) class method), 30–32
[terminate\(\)](#) ([privex.rpcemulator.base.Emulator](#) method), 20, 21, 27
[terminate\(\)](#) ([privex.rpcemulator.bitcoin.BitcoinEmulator](#) method), 16
[test_getblockchaininfo\(\)](#) ([tests.test_bitcoin.TestBitcoinEmulator](#) method), 30, 31, 33
[test_getnetworkinfo\(\)](#) ([tests.test_bitcoin.TestBitcoinEmulator](#) method), 30, 31, 33
[test_getnewaddress\(\)](#) ([tests.test_bitcoin.TestBitcoinEmulator](#) method), 30, 31, 33
[test_send_valid\(\)](#) ([tests.test_bitcoin.TestBitcoinEmulator](#) method), 30, 31, 33
[TestBitcoinEmulator](#) (class in [tests.test_bitcoin](#)), 30, 32
[tests](#) (module), 28
[tests.test_bitcoin](#) (module), 30
[timeout](#) ([privex.rpcemulator.base.QuietRequestHandler](#) attribute), 27

U

[use_coverage](#) ([privex.rpcemulator.base.Emulator](#) attribute), 20, 22, 27

V

[version_string\(\)](#) ([privex.rpcemulator.base.QuietRequestHandler](#) method), 26

W

[wbufsize](#) ([privex.rpcemulator.base.QuietRequestHandler](#) attribute), 26

attribute), [27](#)
weekdayname (*privex.rpcemulator.base.QuietRequestHandler*
attribute), [27](#)