verge-python3 Documentation

Release 0.1.3

Witchspace

Jan 08, 2019

Contents

`verge-python` is a set of Python libraries that allows easy access to the verge peer-to-peer cryptocurrency client API.

Contents:

CHAPTER 1

Getting Started

1.1 Introduction

The goal of this library is to make it easier for:

- Payment gateways to support verge
- Merchant sites to integrate verge payments directly
- Other services that require (micro-)payments to use verge

In this initial release it implements a thin wrapper around the VERGE JSON-RPC API. Using this API from Python directly is conceptually very simple, here is the example from the API documentation page:

```
from jsonrpc import ServiceProxy

access = ServiceProxy("http://user:password@127.0.0.1:20102")
access.getinfo()
access.listreceivedbyaddress(6)
access.sendtoaddress("11yEmxiMso2RsFVfBcCa616npBvGgxiBX", 1000)
```

However, this approach has some disadvantages, one thing is that error handling is complex, as it requires manually checking the contents of `JSONException` objects.

`verge-python` attempts to create an even more friendly interface by wrapping the JSON-RPC API. The major advantages compared to a raw `jsonrpc` based approach are:

- Better exception handling. Exceptions are converted to subclasses of *`VERGEException`*.
- Automatic verge configuration loading. In case the `verge -server` or `verged` program runs on the same machine as the client script, and as the same user, the configuration file can automatically be parsed. This makes it unneccesary to explicitly specify a *username* and *password*. Of course, this is still possible.
- Documentation in Pythonish format. You are reading this right now.
- The functions `getinfo()`, `listreceivedbyaccount()`, `listreceivedbyaddress()`, `listtransactions()` and more return actual Python objects, instead of simply dictionaries. This

makes for cleaner code, as the fields can simply be addressed with `x.foo` instead of `x['foo']`.

The plan for future releases is to add a more high-level interface on top of this.

1.2 Usage

See also the **'main verge documentation'_** for details and background on setting up and using verged remotely.

1.2.1 Setting up verge for remote control

If you run VERGE with the `-server` argument, or if you run `verged`, it can be controlled either by sending it HTTP-JSON-RPC commands.

However, beginning with VERGE 0.3.3 you must create a `VERGE.conf` file in the VERGE data directory (default `$HOME/.vergeconf`) and set an RPC password:

```
rpcuser=anything
rpcpassword=anything
```

Once that is done, the easiest way to check whether VERGE accepts remote commands is by running VERGE again, with the command (and any parameters) as arguments. For example:

```
$ verged getinfo
```

1.2.2 Connecting to the wallet from Python

There are two functions for this:

Connecting to local verge instance Use the function *connect_to_local()*. This automagically sorts out the connection to a verge process running on the current machine, for the current user.

```
conn = vergerpc.connect_to_local()
```

Connecting to a remote verge instance Use the function *connect_to_remote()*. For this function it is necessary to explicitly specify a hostname and port to connect to, and to provide user credentials for logging in.

```
conn = vergerpc.connect_to_remote('foo', 'bar', host='payments.yoyodyne.com',␣
↪port=20102)
```

1.2.3 How to use the API

For basic sending and receiving of payments, the four most important methods are

Getting the current balance Use the method `getbalance()` to get the current server balance.

```
print "Your balance is %f" % (conn.getbalance(),)
```

Check a customer address for validity and get information about it This can be done with the method `validateaddress()`.

```
rv = conn.validateaddress(foo)
if rv.isvalid:
    print "The address that you provided is valid"
else:
    print "The address that you provided is invalid, please correct"
```

Sending payments The method `sendtoaddress()` sends a specified amount of coins to a specified address.

```
conn.sendtoaddress("msTGAm1ApjEJfsWfAaRVaZHRm26mv5GL73", 10000.0)
```

Get a new address for accepting payments To accept payments, use the method `getnewaddress()` to generate a new address. Give this address to the customer and store it in a safe place, to be able to check when the payment to this address has been made.

```
pay_to = conn.getnewaddress()
print "We will ship the pirate sandwidth after payment of 200 coins to ", pay_to
```

Check how much has been received at a certain address The method `getreceivedbyaddress()` returns how many verges have been received at a certain address. Together with the previous function, this can be used to check whether a payment has been made by the customer.

```
amount = conn.getreceivedbyaddress(pay_to)
if amount > 20000.0:
    print "Thanks, your order will be prepared and shipped."
```

1.2.4 The account API

More advanced usage of verge allows multiple accounts within one wallet. This can be useful if you are writing software for a bank, or simply want to have a clear separation between customers payments.

For this, see the Account API documentation.

1.3 Examples

A basic program that uses `python-verge` looks like this:

First, import the library and exceptions.

```
import vergerpc
from vergerpc.exceptions import InsufficientFunds
```

Then, we connect to the currently running `verge` instance of the current user on the local machine with one call to *`connect_to_local()`*. This returns a `VERGEConnection` objects:

```
conn = vergerpc.connect_to_local()
```

Try to move one verge from account `testaccount` to account `testaccount2` using `move()`. Catch the *`InsufficientFunds`* exception in the case the originating account is broke:

```
try:
    conn.move("testaccount", "testaccount2", 100.0)
except InsufficientFunds,e:
    print "Account does not have enough funds available!"
```

Retrieve general server information with `getinfo()` and print some statistics:

```
info = conn.getinfo()
print "Blocks: %i" % info.blocks
print "Connections: %i" % info.connections
print "Difficulty: %f" % info.difficulty
```

CHAPTER 2

API reference

2.1 vergerpc — Convenience functions

verge-python - Easy-to-use VERGE API client

vergerpc.**connect_to_local**(*filename=None*)
Connect to default verge instance owned by this user, on this machine.

Returns a VERGEConnection object.

Arguments:

- *filename*: Path to a configuration file in a non-standard location (optional)

vergerpc.**connect_to_remote**(*user*, *password*, *host='localhost'*, *port=20102*, *use_https=False*)
Connect to remote or alternative local verge client instance.

Returns a VERGEConnection object.

2.2 vergerpc.connection — Connect to VERGE server via JSON-RPC

2.3 vergerpc.exceptions — Exception definitions

Exception definitions.

exception vergerpc.exceptions.**ClientException**(*error*)
Bases: *vergerpc.exceptions.VERGEException*

P2P network error. This exception is never raised but functions as a superclass for other P2P client exceptions.

exception vergerpc.exceptions.**DownloadingBlocks**(*error*)
Bases: *vergerpc.exceptions.ClientException*

Client is still downloading blocks.

exception vergerpc.exceptions.**InsufficientFunds**(*error*)
Bases: *vergerpc.exceptions.WalletError*

Insufficient funds to complete transaction in wallet or account

exception vergerpc.exceptions.**InvalidAccountName**(*error*)
Bases: *vergerpc.exceptions.WalletError*

Invalid account name

exception vergerpc.exceptions.**InvalidAddressOrKey**(*error*)
Bases: *vergerpc.exceptions.VERGEException*

Invalid address or key.

vergerpc.exceptions.**InvalidAmount**
alias of *vergerpc.exceptions.JSONTypeError*

exception vergerpc.exceptions.**InvalidParameter**(*error*)
Bases: *vergerpc.exceptions.VERGEException*

Invalid parameter provided to RPC call.

vergerpc.exceptions.**InvalidTransactionID**
alias of *vergerpc.exceptions.InvalidAddressOrKey*

exception vergerpc.exceptions.**JSONTypeError**(*error*)
Bases: *vergerpc.exceptions.VERGEException*

Unexpected type was passed as parameter

exception vergerpc.exceptions.**KeypoolRanOut**(*error*)
Bases: *vergerpc.exceptions.WalletError*

Keypool ran out, call keypoolrefill first

exception vergerpc.exceptions.**NotConnected**(*error*)
Bases: *vergerpc.exceptions.ClientException*

Not connected to any peers.

exception vergerpc.exceptions.**OutOfMemory**(*error*)
Bases: *vergerpc.exceptions.VERGEException*

Out of memory during operation.

exception vergerpc.exceptions.**SafeMode**(*error*)
Bases: *vergerpc.exceptions.VERGEException*

Operation denied in safe mode (run `verged` with `-disablesafemode`).

vergerpc.exceptions.**SendError**
alias of *vergerpc.exceptions.WalletError*

exception vergerpc.exceptions.**TransportException**(*msg*, *code=None*, *protocol=None*, *raw_detail=None*)
Bases: `exceptions.Exception`

Class to define transport-level failures.

exception vergerpc.exceptions.**VERGEException**(*error*)
Bases: `exceptions.Exception`

Base class for exceptions received from VERGE server.

- *code* – Error code from `verged`.

exception vergerpc.exceptions.**WalletAlreadyUnlocked**(*error*)
Bases: *vergerpc.exceptions.WalletError*

Wallet is already unlocked

exception vergerpc.exceptions.**WalletEncryptionFailed**(*error*)
Bases: *vergerpc.exceptions.WalletError*

Failed to encrypt the wallet

exception vergerpc.exceptions.**WalletError**(*error*)
Bases: *vergerpc.exceptions.VERGEException*

Unspecified problem with wallet (key not found etc.)

exception vergerpc.exceptions.**WalletPassphraseIncorrect**(*error*)
Bases: *vergerpc.exceptions.WalletError*

The wallet passphrase entered was incorrect

exception vergerpc.exceptions.**WalletUnlockNeeded**(*error*)
Bases: *vergerpc.exceptions.WalletError*

Enter the wallet passphrase with walletpassphrase first

exception vergerpc.exceptions.**WalletWrongEncState**(*error*)
Bases: *vergerpc.exceptions.WalletError*

Command given in wrong wallet encryption state (encrypting an encrypted wallet etc.)

vergerpc.exceptions.**wrap_exception**(*error*)
Convert a JSON error object to a more specific VERGE exception.

2.4 vergerpc.data — VERGE RPC service, data objects

VERGE RPC service, data objects.

class vergerpc.data.**AccountInfo**(**args_t*, ***args_d*)
Bases: vergerpc.util.DStruct

Information object returned by `listreceivedbyaccount()`.

- *account* – The account of the receiving address.
- *amount* – Total amount received by the address.
- *confirmations* – Number of confirmations of the most recent transaction included.

class vergerpc.data.**AddressInfo**(**args_t*, ***args_d*)
Bases: vergerpc.util.DStruct

Information object returned by `listreceivedbyaddress()`.

- *address* – Receiving address.
- *account* – The account of the receiving address.
- *amount* – Total amount received by the address.
- *confirmations* – Number of confirmations of the most recent transaction included.

class vergerpc.data.**AddressValidation**(**args_t*, ***args_d*)
Bases: vergerpc.util.DStruct

Information object returned by `validateaddress()`.

- *isvalid* – Validatity of address (`True` or `False`).
- *ismine* – `True` if the address is in the server's wallet.
- *address* – VERGE address.

class vergerpc.data.**MiningInfo**(**args_t*, ***args_d*)
Bases: vergerpc.util.DStruct

Information object returned by `getmininginfo()`.

- *blocks* – Number of blocks.
- *currentblocksize* – Size of current block.
- *currentblocktx* – Number of transactions in current block.
- *difficulty* – Current generating difficulty.
- *errors* – Number of errors.
- *generate* – True if generation enabled, False if not.
- *genproclimit* – Processor limit for generation.
- *hashespersec* – Number of hashes per second (if generation enabled).
- *pooledtx* – Number of pooled transactions.
- *testnet* – True if connected to testnet, False if on real network.

class vergerpc.data.**ServerInfo**(**args_t*, ***args_d*)
Bases: vergerpc.util.DStruct

Information object returned by `getinfo()`.

- *errors* – Number of errors.
- *blocks* – Number of blocks.
- *paytxfee* – Amount of transaction fee to pay.
- *keypoololdest* – Oldest key in keypool.
- *genproclimit* – Processor limit for generation.
- *connections* – Number of connections to other clients.
- *difficulty* – Current generating difficulty.
- *testnet* – True if connected to testnet, False if on real network.
- *version* – VERGE client version.
- *proxy* – Proxy configured in client.
- *hashespersec* – Number of hashes per second (if generation enabled).
- *balance* – Total current server balance.
- *generate* – True if generation enabled, False if not.
- ***unlocked_until* – Timestamp (seconds since epoch) after which the wallet** will be/was locked (if wallet encryption is enabled).

class vergerpc.data.**TransactionInfo**(**args_t*, ***args_d*)
Bases: vergerpc.util.DStruct

Information object returned by `listtransactions()`.

- *account* – account name.
- *address* – the address verges were sent to, or received from.
- *category* – will be generate, send, receive, or move.
- *amount* – amount of transaction.
- *fee* – Fee (if any) paid (only for send transactions).
- *confirmations* – number of confirmations (only for generate/send/receive).
- *txid* – transaction ID (only for generate/send/receive).
- *otheraccount* – account funds were moved to or from (only for move).
- *message* – message associated with transaction (only for send).
- *to* – message-to associated with transaction (only for send).

class vergerpc.data.**WorkItem**(**args_t*, ***args_d*)
Bases: vergerpc.util.DStruct

Information object returned by `getwork()`.

- *midstate* – Precomputed hash state after hashing the first half of the data.
- *data* – Block data.
- *hash1* – Formatted hash buffer for second hash.
- *target* – Little endian hash target.

2.5 vergerpc.config — Utilities for reading verge configuration files

Utilities for reading verge configuration files.

vergerpc.config.**read_config_file**(*filename*)
Read a simple `'='`-delimited config file. Raises `IOError` if unable to open file, or `ValueError` if an parse error occurs.

vergerpc.config.**read_default_config**(*filename=None*)
Read verge default configuration from the current user's home directory.

Arguments:

- *filename*: Path to a configuration file in a non-standard location (optional)

CHAPTER 3

Testing

The easiest way to run the tests is to use:

```
$ python tests/test.py
```

This will read the *~/.VERGE/VERGE.conf* configuration file and connect to the local *VERGEd* instance.

The *–config* option can be used to read a different local configuration file, you probably want to use the *–nolocal* option as well. This is useful if you want to use a separate verge data directory for development purposes.

3.1 Configuration from environment variables

This works independently from the previous methods. The recommended way to use this is to install the bitcoin testnet box. After a *make start* inside the box you should be able to run the tests with:

```
HOST=localhost USER1=admin1 PORT1=20102 USER2=admin2 PORT2=21102 PASS=123 python␣
↪tests/test.py --envconfig
```

This was primarily added to make continuous integration with travis-ci possible. The environment variables in the *.travis.yml* file are encrypted as described in http://about.travis-ci.org/docs/user/build-configuration/#Secure-environment-variables and use a testnetbox outside of travis to avoid installing the testnet box all the time.

CHAPTER 4

Indexes and tables

- genindex
- modindex
- search

Python Module Index

v

Index