BarterDEX Documentation

Release 0.1a2

Jorian , gcharang , Contributors from Komodo

Jul 05, 2018

User documentation

BarterDEX is a decentralized cryptocurrency exchange using atomic swaps to trade any coin A directly with another coin B, without needing to trust a third party. Users always maintain control of their private keys.

CHAPTER 1

Frequently Asked Questions

1.1 What is an atomic swap?

An Atomic Swap is an on-chain, direct exchange of 2 different coins. For example KMD <-> LTC or BTC <-> SYS. It is atomic, because in a decentralized exchange, both parties must be assured that the other party is not able to skip out on his part, resulting in a loss of funds for 1 party.

Atomic means that a swap either totally succeeds or not at all.

Certain safeguards are needed to make sure the stealing of funds is not possible by either party.

1.2 How to get listed on BarterDEX?

The requirements for a coin to be able to do an atomic swap are:

- have BIP65 (Check LockTime Verify) implemented;
- support the following standard Bitcoin API methods (http://chainquery.com/bitcoin-api)

```
estimatefee                   importaddress
getblock                      importprivkey
getblockhash                  listunspent
getinfo                       listreceivedbyaddress
getrawmempool                 listtransactions
getrawtransaction             validateaddress
gettxout                      sendrawtransaction
                              signrawtransaction
```

Note that apart from supporting the above requirements, no additional custom code is needed to be able to trade on BarterDEX. (This in contrast to the misinformation being spread on several forums, where is stated that BarterDEX requires customizations to coins)

1.3 Electrum or Native?

Electrum lets you use the blockchain of the coin you want to trade, without having to download, install and sync that wallet. Just select Electrum and you are good to go.

Electrum is a Simple Payment Verification (SPV) service, somewhere on a server, that can be used to verify transactions on the network of the selected coin, without having to expose your private key. This means you can use this SPV service instead of going through the hassle of downloading the wallet of your coin, synchronizing the blockchain etc.

Native is when you download, install and sync the wallet of the coin you want to trade. Trading with native wallets is faster and more stable than using Electrum. It is highly recommended for Liquidity Providers to use Native and not to use Electrum.

1.4 What are Zcredits?

Zcredits are credits for being able to do zero confirmation swaps. Zero confirmation swaps let you swap 2 coins in approximately 30 seconds, instead of waiting for the coin's block confirmation times.

You get Zcredits by depositing KMD and locking it for an X amount of weeks. So if you deposit 10 KMD you get 10 Zcredits, and trades worth up to 10 KMD are now eligible for Zero Confirmation swaps. This is valid for all coins, even if KMD is not involved in a trade. BarterDEX will determine the value of the trade equivalent in KMD to check if a Zero Confirmation swap is possible. For BarterDEX to be able to determine this, a price must be set for the coin/KMD pair.

1.5 What are UTXOs?

UTXO stands for Unspent Transaction Output. BarterDEX trades UTXOs, not balances. This makes trading different from trading on a centralized exchange. Because of the atomic swap protocol, UTXOs must have certain sizes to be eligible for a trade.

Basically this means that you need to make at least 3 transactions to your smartaddress to be able to trade. These transactions should have sizes X, X * 1.2 and X * 0.1.

1.6 Why are multiple UTXOs needed?

BarterDEX is an UTXO based exchange. This means that 1 UTXO is exchanged for 1 other UTXO. For example: 1 KMD UTXO with a value of 32 KMD is traded for 1 BCH UTXO with a value of 0.5 BCH.

1.7 Why can't I claim my expired 0-conf deposit?

This is due to avoid bad actors stealing the deposit, and depends on the time you made a deposit. Funds are available 3 - 10 days after the expiration of the number of weeks you defined when making the deposit.

The calculation in the code:

```
timestamp = (uint32_t) time(NULL);
timestamp /= LP_WEEKMULT;
timestamp += weeks + 2;
timestamp *= LP_WEEKMULT;
```

Doing a claim on a still locked deposit will return the time you need to wait before your deposit can successfully be claimed.

1.8 Who are alice and bob?

In BarterDEX terms, alice is the one who initiates a trade. She buys an order from bob, who already put up an order in the orderbook. It is like saying bob has a marketstand with something to sell and alice walks up to the stand to buy something.

1.9 Is it free to cancel an order?

Yes. Placing orders and sending a request doesn't cost you anything. Only when your request has found a willing trade partner and a connection has been established, the dexfee and transaction fees will be paid.

1.10 How do I get the private key of my smartaddress?

BarterDEX uses watch-only addresses, which basically means that BarterDEX is a trade wallet. The passphrase you enter when starting BarterDEX is the access to your trade wallet and thus, coins.

It requires starting `marketmaker` from the command line to retrieve all private keys of all the smartaddresses, based on the passphrase. You do this by adding `"wif":1` to the marketmaker startup arguments json. In the initial `getcoin` that marketmaker does, it will return all wifs for each smartaddress.

For Komodo formatted addresses, it is possible to do the *calcaddress* api call.

1.11 How much are the fees?

Fees for using the exchange exist in paying a dexfee, to be paid by alice (the one initiating the trade), also called the taker fee. This is about 0.15% of the alicepayment - the amount you're sending to the other party.

There are no maker fees.

You also pay the standard transaction fees, for sending the payment to the other party.

The dexfees are collected in a specific address, generated from the rmd160 hash ca1e04745e8ca0c60d8c5881531d51bec470743f. For KMD, this is RThtXup6Zo7LZAi8kRWgjAyi1s4u6U9Cpf, BTC: 1KRhTPvoxyJmVALwHFXZdeeWFbcJSbkFPu. Once a significant amount of fees are collected, the fees are paid as dividend to DEX assetholders. DEX is a Komodo asset and can be traded on BarterDEX.

1.12 Currently supported coins

Coin	Name	Asset	Name/description
BTC	Bitcoin	REVS	Revenue Shares
LTC	Litecoin	SUPERNET	Supernet / Unity
KMD	Komodo	DEX	InstantDEX
BTG	Bitcoin Gold	PANGEA	Pangea Poker
BCH	Bitcoin Cash	JUMBLR	JUMBLR
ZEC	Zcash	BET	BET Platform
VTC	VertCoin	CRYPTO	CRYPTO777
DOGE	DogeCoin	HODL	HODL
HUSH	Hush	MSHARK	MSHARK
GRS	GroestlCoin	BOTS	Tradebots
DGB	DigiByte	COQUI	Coqui
XMCC	Monoeci	WLC	WirelessCoin
BTCH	Bitcoin Hush	KV	Key-Value
CRC	CrowdCoin	CEAL	CEAL
VOT	VoteCoin	MESH	MESH
INN	Innova	ETOMIC	ERC20
MOON	MoonCoin		
CRW	Crown		
EFL	eGulden		
GBX	GoByte		
BCO	BridgeCoin		
BLK	BlackCoin		
ABY	Applebyte		
STAK	Straks		
XZC	Zcoin		
QTUM	QTUM		

Continued on next page

Table 1 – continued from previous page

Coin	Name	Asset	Name/description
PURA	PURA		
DSR	Desire		
MNZ	Monaize		
BTCZ	Bitcoin Z		
MAGA	MagaCoin		
BSD	Bitsend		
IOP	IoP		
BLOCK	BlockNET DX		
CHIPS	CHIPS		
888	OctoCoin		
ARG	Argentum		
GLT	Global Token		
ZER	Zero		
HODLC	HOdlcoin		
UIS	Unitus		

All the Komodo Platform assetchains

1.13 What are the differences between BarterDEX and BlockNET DX?

BlockNET DX, or BlockDX, is a coin which focus solely lies on creating a Decentralized Exchange. The differences between BarterDEX and BlockDX are subtle, but important.

1. First of all, the fees:

fees:	BlockDX	BarterDEX
Maker-fee	0.05%	none
Taker-fee	0.20%	0.15%

These fees do not take standard transaction fees into account, which for BarterDEX is based on the amount of transactions necessary to do an atomic swap (4 for maker, 3 for taker). BlockDX is (at the time of this writing) not yet live, so nothing can be said of how many standard transaction fees they require.

2. BlockDX uses so-called Service Nodes to be able to do atomic swaps. They are essential in the process of doing an atomic swap; without the Service Nodes, you can't trade. This in stark contrast with BarterDEX, where you don't need Service Nodes to be able to trade. 2 nodes, both running BarterDEX, are able to do an atomic swap between each other, without needing anything else.

3. To use the BlockDX, you need to download and install the BlockDX wallet. BarterDEX is not tied to any cryptocurrency; all you need is the marketmaker executable that gives you access to BarterDEX networks.

At the time of this writing, BlockDX has yet to release (a beta of) their DEX and its source code. Until then, no proper comparison can be made between BlockDX and BarterDEX. It is clear though, that BlockDX is going to be the closest 'competitor' to BarterDEX.

1.14 Do I need to leave BarterDEX running all the time?

Yes. Atomic swaps needs transactions signed with your private key, so you need to leave BarterDEX running to be able to execute orders.

Yes, that possibility exists, but for now it's only done using the Command Line. See the guide in our Guides section explaining what needs to be done.

CHAPTER 2

Getting Started with BarterDEX

2.1 What you need to know before using BarterDEX

differences between BarterDEX and a centralized exchange coin confirmation times

UTXO trading: 2+ deposits minimum in order to trade. account for network fees Total balance is sum of utxos. change gets back into smartaddress, creating a new UTXO.

alice and bob, which payments need to happen in order to atomically swap

buying an order (mini auction story) request takes at least 60 seconds

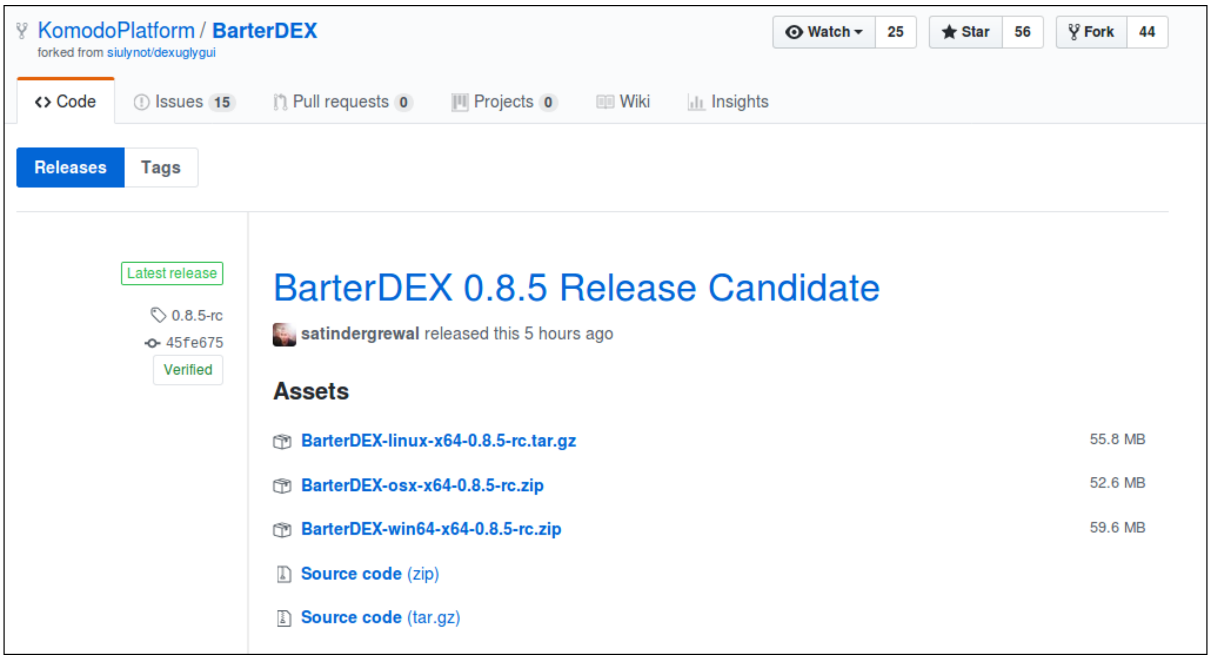
How does the decentralized orderbook work, p2p etc. not always uptodate

setting up the native wallet + .conf file -> for each coin?

2.2 Windows

2.2.1 Download and setup

Download the latest version of BarterDEX



BarterDEX 0.8.5 Release Candidate

satindergrewal released this 5 hours ago

Assets

BarterDEX-linux-x64-0.8.5-rc.tar.gz	55.8 MB
BarterDEX-osx-x64-0.8.5-rc.zip	52.6 MB
BarterDEX-win64-x64-0.8.5-rc.zip	59.6 MB
Source code (zip)	
Source code (tar.gz)	

Create a folder on your desktop and drag the win64 folder in it:

CHAPTER 3

Guides

3.1 How to use BarterDEX using CLI

Using BarterDEX with the Command Line Interface, you get access to all the low level Remote Procedure Calls (RPC) that exist in BarterDEX. It gives you power and control over how you want to be a trader / marketmaker. Throughout this guide, the command line (terminal) is used for installing BarterDEX.

3.1.1 Requirements

At this time, only Linux and MacOS are supported.

Linux (16.04)

Download and install Nanomsg:

```
git clone https://github.com/nanomsg/nanomsg
cd nanomsg
cmake .
make
sudo make install
sudo ldconfig
```

Install required packages:

```
sudo apt-get update && sudo apt-get install git libcurl4-openssl-
↪dev build-essential
```

MacOS

To install Nanomsg on MacOS, it's easiest to install it using Homebrew. Homebrew is a handy package manager for MacOS. If you don't have it already, download it by entering the following into a Terminal window:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
↪Homebrew/install/master/install)"
```

Afterwards, install Homebrew by typing:

```
brew install nanomsg
```

That's it!

3.1.2 Installation & Setup

Installation is the same for Linux and MacOS. Start by cloning the SuperNET repository:

```
git clone https://github.com/jl777/SuperNET -b dev
```

This will download all the files necessary to start using BarterDEX from the command line. It uses the Development branch, assuring you with the latest updates. If you don't want this, type `git checkout master` to work with more stable releases. If you want the very latest, untested changes, type `git checkout jl777`.

Now navigate to the SuperNET folder and execute the install script:

```
cd ~/SuperNET/iguana/exchanges
./install
```

This copies a bunch of scripts in the dexscripts folder, which is where we need to go next. This dexscripts folder now contains executable scripts that issue the most common API calls.

```
cd ../dexscripts
```

Before we can use any of these scripts, some security measures are needed to prevent bad actors from issuing API calls without your consent. The `passphrase` and `userpass` values take care of this security.

Create the passphrase file. Type the following and hit enter:

```
nano passphrase
```

This file should contain the following line, with a strong passphrase between the "":

```
export passphrase="<strong userpass value here>"
```

`Ctrl-x` to exit, press `y` and then `enter` to save the changes.

The userpass value is derived from the passphrase value, and in order to obtain the userpass value, we need to start BarterDEX. Starting BarterDEX (or actually the marketmaker process) is done by executing the client script, which basically is an automated combination of retrieving the latest updates and building the marketmaker executable file: (it can take a while before anything shows up)

```
./client
```

Let it load until you see a line that starts with >>>>>>>>>> DEX stats 127.0.0.1:7783. This means the BarterDEX node is now up and running and that it's able to listen for commands.

To obtain the userpass value, you need to check the output in the terminal. Somewhere along these lines, you'll find: userpass.(a_long_line_of_random_numbers_and_letters) Copy this value and paste it in a newly created userpass file:

```
nano userpass
export userpass="<paste userpass value here>"
```

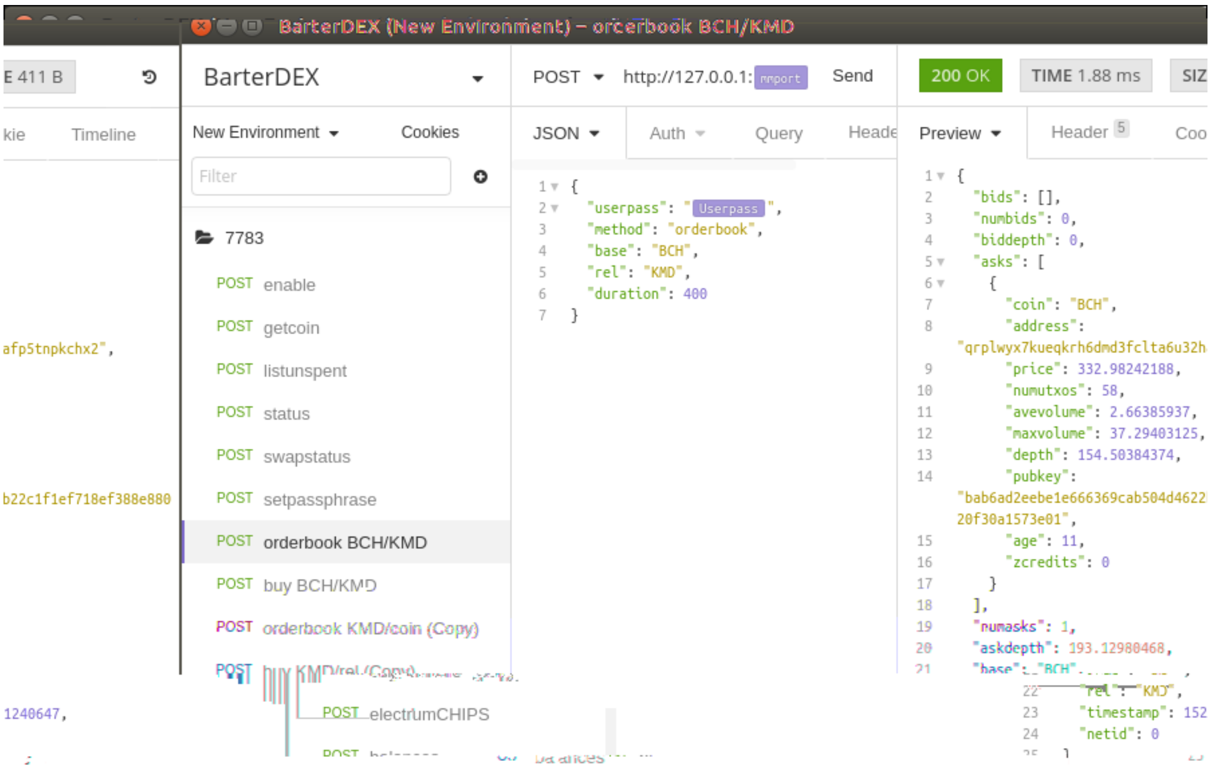
Ctrl-x to exit, y and enter to save changes.

Everything is now good to go. From here on, you can issue any script that is in the dexscripts folder, such as the orderbook script, that fetches all the orders from the specified pair, or the getcoin script that gets all the coin-specific information from the coin as defined inside that script.

The API docs explain all the BarterDEX API calls.

3.2 How to use Insomnia together with the CLI

Insomnia is a great tool to replace the terminal window, but still be able to issue all the API calls in a visually more attractive way. Insomnia stores all the different API calls in a folder structure, with the possibility to set environment variables on all calls in a folder. This makes it quite easy to maintain different netid's or manage multiple marketmakers on the same netid using a different RPC port.



```
{
  "userpass": "Userpass",
  "method": "orderbook",
  "base": "BCH",
  "rel": "KMD",
  "duration": 400
}
```

```
{
  "bids": [],
  "numbids": 0,
  "biddepth": 0,
  "asks": [
    {
      "coin": "BCH",
      "address":
"qrplwyx7kueqkrh6dmd3fclta6u32h
      "price": 332.98242188,
      "numutxos": 58,
      "avevolume": 2.66385937,
      "maxvolume": 37.29403125,
      "depth": 154.50384374,
      "pubkey":
"bab6ad2eebe1e666369cab504d4622
20f30a1573e01",
      "age": 11,
      "zcredits": 0
    }
  ],
  "numasks": 1,
  "askdepth": 193.12980468,
  "base": "BCH",
  "rel": "KMD",
  "timestamp": 152
  "netid": 0
```

Download Insomnia here: https://insomnia.rest. It is available for all platforms. This guide will use Linux as an example.

It is not necessary to have native coin daemons running. As you will see, managing different coins using Electrum in Insomnia is not difficult.

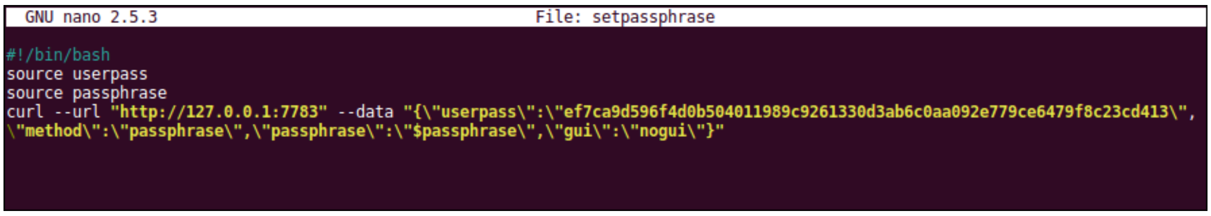
Make sure to follow the above guide: *How to use BarterDEX using CLI*. We need to be able to start a marketmaker instance from the command line in order to start using Insomnia.

3.2.1 Setting the passphrase

The first call you need to do when you start a marketmaker instance, is to set the passphrase using the `setpassphrase` call. Normally, using a CLI, you go to `~/SuperNET/iguana/dexscripts` and execute the `setpassphrase` script stored there. Now, we are going to open that script and copy the contents to Insomnia.

Go to the `dexscripts` folder:

```
cd ~/SuperNET/iguana/dexscripts
nano setpassphrase
```



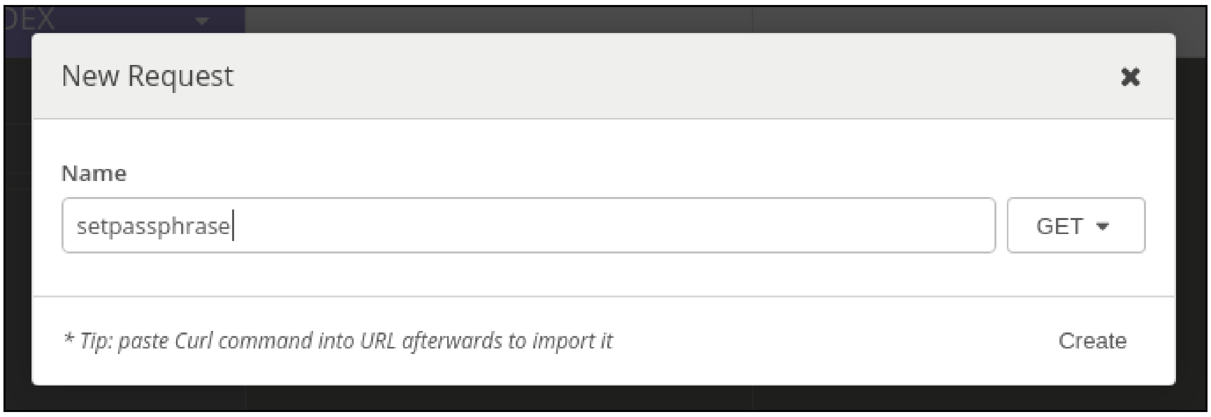
```
  GNU nano 2.5.3                                   File: setpassphrase

#!/bin/bash
source userpass
source passphrase
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"ef7ca9d596f4d0b504011989c9261330d3ab6c0aa092e779ce6479f8c23cd413\",
\"method\":\"passphrase\",\"passphrase\":\"$passphrase\",\"gui\":\"nogui\"}"
```

You'll see that this script uses the passphrase as defined in the passphrase file, and that the curl command below it is the RPC issued to the marketmaker instance. It is only this curl command we need in Insomnia.

Copy the following `setpassphrase` curl command to your clipboard (it is the same as the one in the dexscripts folder):

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\
↪"ef7ca9d596f4d0b504011989c9261330d3ab6c0aa092e779ce6479f8c23cd413\
↪",\"method\":\"passphrase\",\"passphrase\":\"$passphrase\",\"gui\
↪":\"nogui\"}"
```

Now, go to Insomnia, and create a New Request (Ctrl-N).

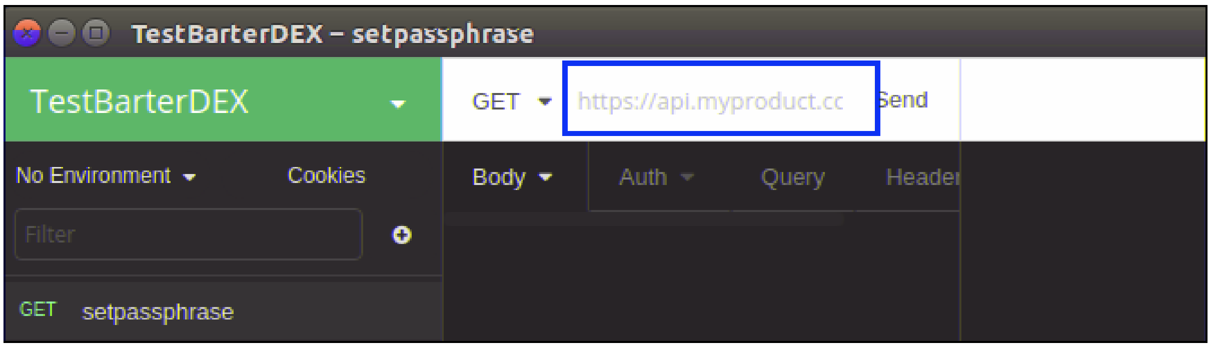


setpassphrase

* Tip: paste Curl command into URL afterwards to import it

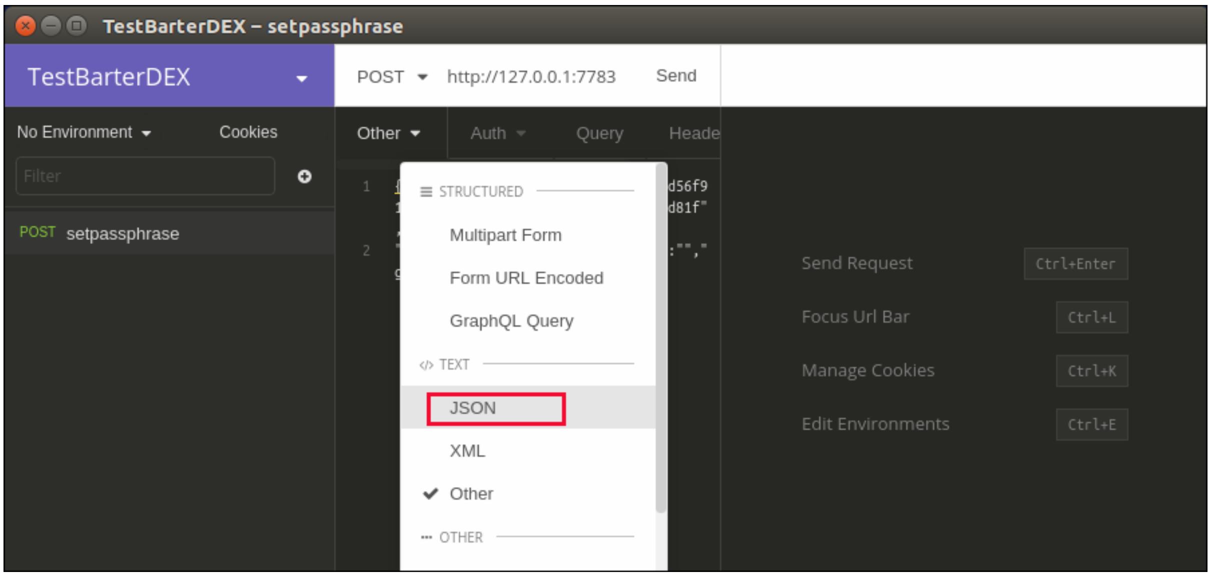
Name it `setpassphrase` and click Create.

Paste the just copied curl command in the textfield area, right next to the the GET dropdown:

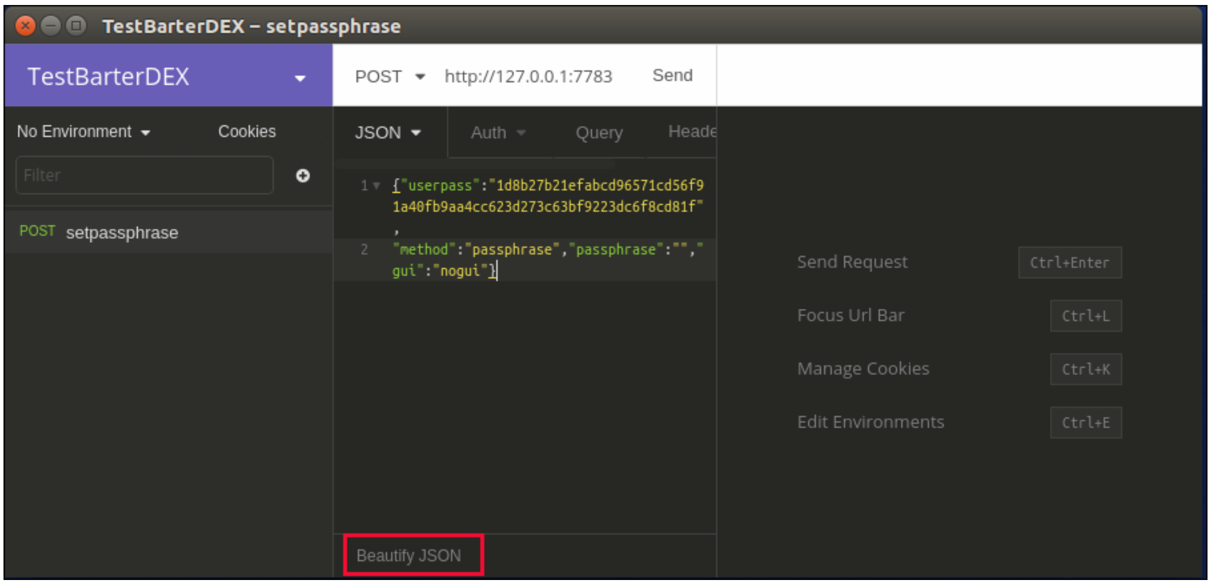


Insomnia recognises this curl command, and automatically extracts the ip-address and the data. Let's call that middle part of Insomnia the input screen. It still looks a bit ugly, so let's make it look better.

Click on Other, and change Other to JSON.

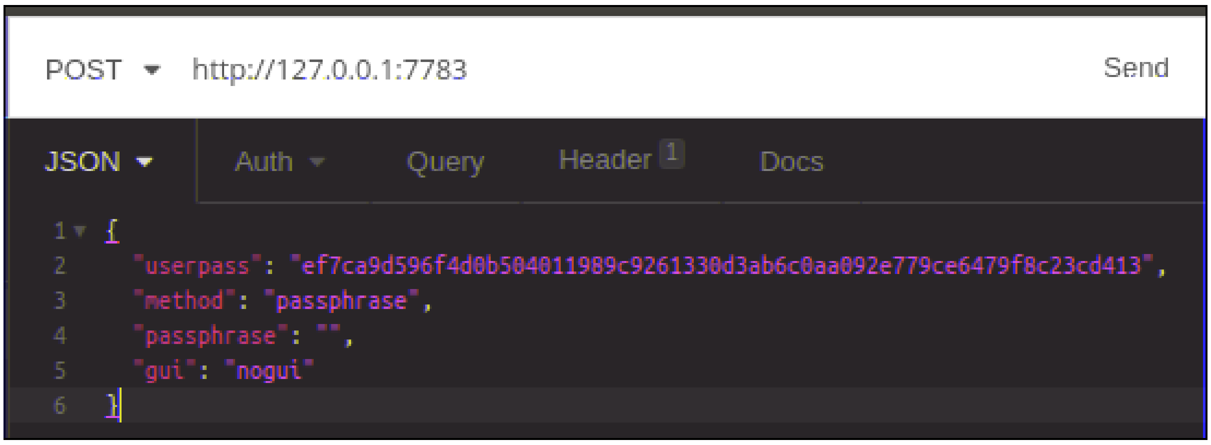


Next, Beautify this JSON:



```
{"userpass":"1d8b27b21efabcd96571cd56f9
1a40fb9aa4cc623d273c63bf9223dc6f8cd81f"
,
"method":"passphrase","passphrase":"","
gui":"nogui"}
```

It should result in this:



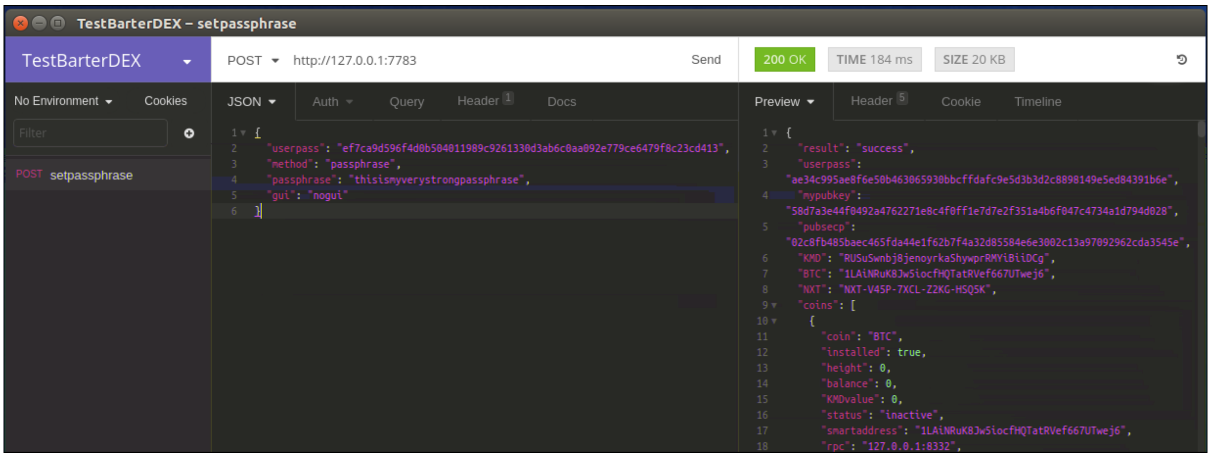
```
{
    "userpass": "ef7ca9d596f4d0b504011989c9261330d3ab6c0aa092e779ce6479f8c23cd413",
    "method": "passphrase",
    "passphrase": "",
    "gui": "nogui"
}
```

Looks better, right? This process of copying a curl command from the `dexscripts` folder, creating a new request and pasting the curl command in Insomnia is what you probably need to do for most of the commands, like `orderbook`, `buy` and `balances`. Actually, all calls as defined in the *API docs* can be copied into Insomnia.

Now enter your passphrase in the area where the passphrase still is empty, between the 2 quotes. Start a marketmaker instance by running `./client` from the *dexscripts* folder and let it boot. When it's done booting, click the Send button in Insomnia for the setpassphrase request.

(if the output on the right side of Insomnia complains that the userpass has not been set, make sure to set the userpass value in the JSON data with `ef7ca9d596f4d0b504011989c9261330d3ab6c0aa092e779ce6479f8c23cd413`).

This is what you should see in the output part of the screen, when you clicked Send:

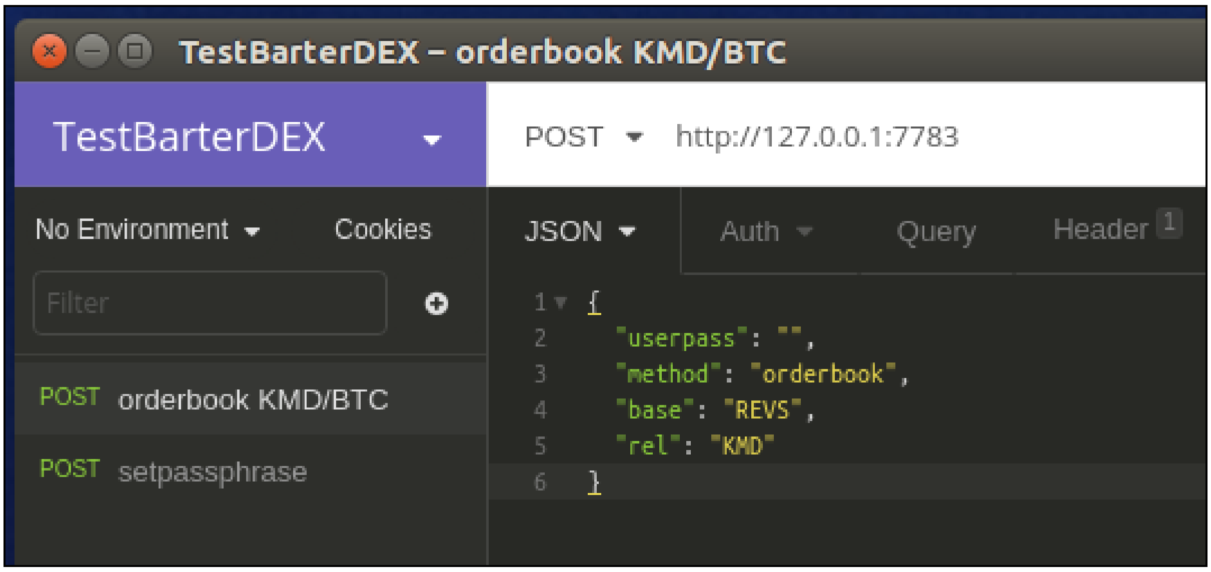


```
{
  "userpass": "ef7ca9d596f4d0b504011989c9261330d3ab6c0aa092e779ce6479f8c23cd413",
  "method": "passphrase",
  "passphrase": "thisismyverystrongpassphrase",
  "gui": "nogui"
}
```

```
{
  "result": "success",
  "userpass":
"ae34c995ae8f6e50b463065930bbcffdafc9e5d3b3d2c8898149e5ed84391b6e",
  "mypubkey":
"58d7a3e44f0492a4762271e8c4f0ff1e7d7e2f351a4b6f047c4734a1d794d028",
  "pubsecp":
"02c8fb485baec465fda44e1f62b7f4a32d85584e6e3002c13a97092962cda3545e",
  "KMD": "RUSuSwnbj8jenoyrkaShywprRMYiBiiDCg",
  "BTC": "1LAiNRuK8Jw5iocfHQTatRVef667UTwej6",
  "NXT": "NXT-V45P-7XCL-Z2KG-HSQ5K",
  "coins": [
    {
      "coin": "BTC",
      "installed": true,
      "height": 0,
      "balance": 0,
      "KMDvalue": 0,
      "status": "inactive",
      "smartaddress": "1LAiNRuK8Jw5iocfHQTatRVef667UTwej6",
      "rpc": "127.0.0.1:8332",
```

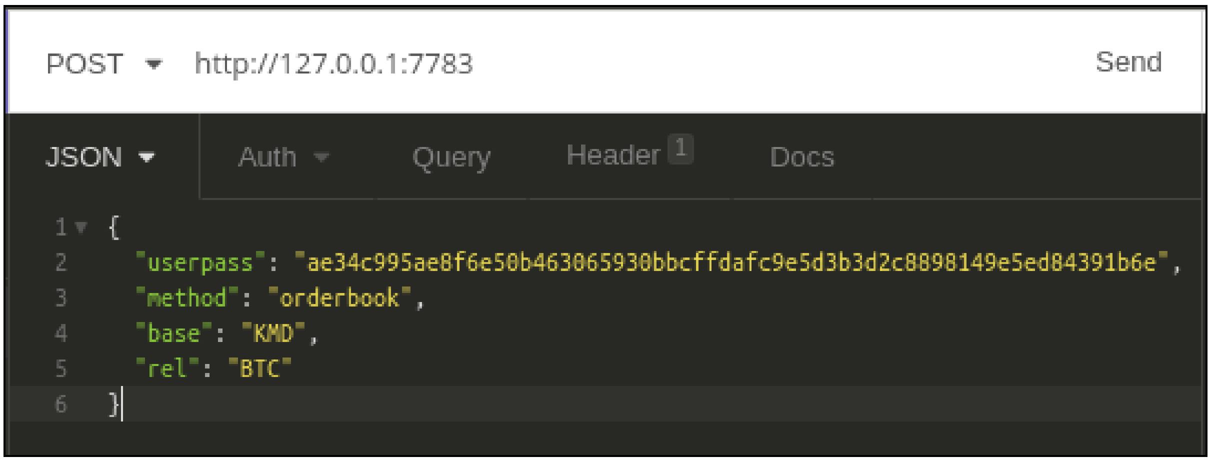
3.2.2 Fetch the orderbook

The next thing you probably want to see, is an orderbook for some pair, like KMD/BTC. Go to the `dexscripts` folder again, copy the complete curl command for `orderbook` and paste it in a new request. I called this new request `orderbook KMD/BTC` and the end result should look like this:



```
{
  "userpass": "",
  "method": "orderbook",
  "base": "REVS",
  "rel": "KMD"
}
```

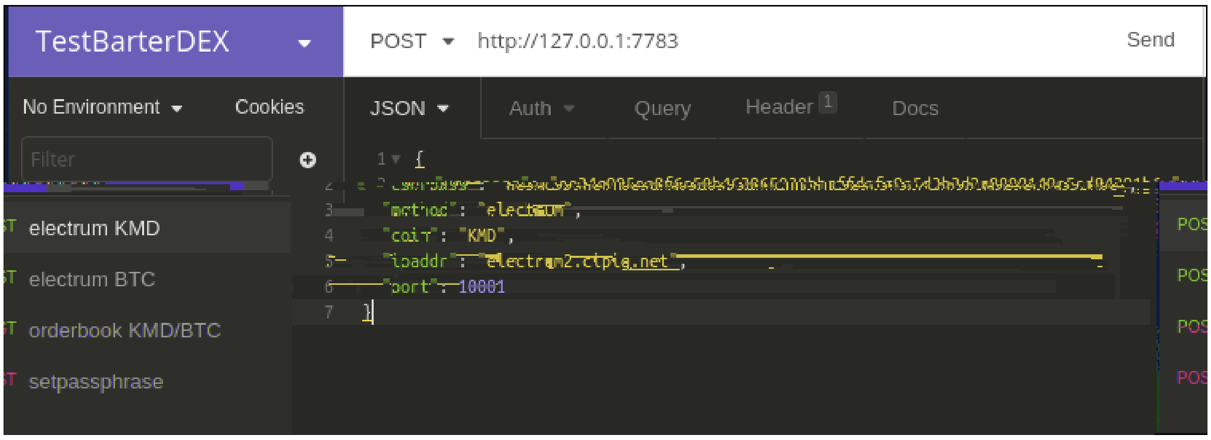
Since with KMD/BTC, you're saying you want to buy KMD with BTC, the data in the JSON needs to be changed. Change REVS to KMD and KMD to BTC, such that `"base": "KMD"` and `"rel": "BTC"`. You also need to copy the userpass from the `setpassphrase` call we did before. On the second line in the output of the `setpassphrase` call, you see a userpass value. Copy this value and paste it in the `orderbook` request. It should end up like this:



```
{
    "userpass": "ae34c995ae8f6e50b463065930bbcffdafc9e5d3b3d2c8898149e5ed84391b6e",
    "method": "orderbook",
    "base": "KMD",
    "rel": "BTC"
}
```

Try Sending this request. It will complain that at least one of the coins is disabled, so we need to enable them. A coin must be explicitly enabled before trades can happen. By default, all coins except KMD and BTC are disabled at startup, which means that if you have a native KMD or BTC daemon running, you don't have to explicitly enable KMD or BTC. If you don't have a KMD daemon, or a BTC daemon, you need to use a Electrum SPV for that. Let's first enable both coins using Electrum.

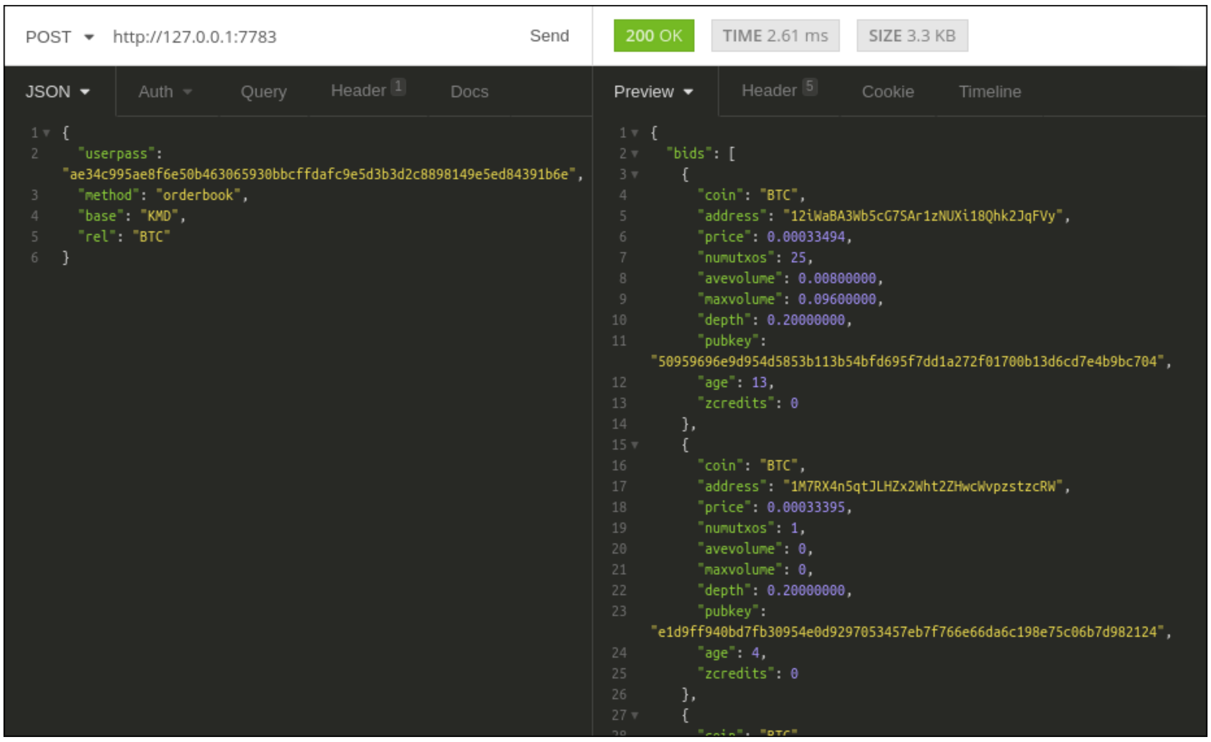
Go to http://pad.supernet.org/electrum-servers where you'll find a long list of all coins that support Electrum (https://github.com/jl777/coins will contain all electrum servers in the future). Find BTC, copy the curl command and paste it in Insomnia, like you did with the other requests. Do the same for KMD in a new request, such that you end up with 2 requests: electrum BTC and electrum KMD:



```
{
    [illegible]
    "method": "electrum",
    "coin": "KMD",
    "ipaddr": "electrum2.cipig.net",
    "port": 10001
}
```

Click Send for both requests, and if all is right, you'll see a success message for both requests in the output screen.

Now that both coins are enabled (a successful electrum request automatically enables the coin), we can go to the orderbook request and see if something happens. If all is right, you'll see something like this:



```
{
  "userpass":
"ae34c995ae8f6e50b463065930bbcffdafc9e5d3b3d2c8898149e5ed84391b6e",
  "method": "orderbook",
  "base": "KMD",
  "rel": "BTC"
}
```

```
{
  "bids": [
    {
      "coin": "BTC",
      "address": "12iWaBA3Wb5cG7SAr1zNUXi18Qhk2JqFVy",
      "price": 0.00033494,
      "numutxos": 25,
      "avevolume": 0.00800000,
      "maxvolume": 0.09600000,
      "depth": 0.20000000,
      "pubkey":
"50959696e9d954d5853b113b54bfd695f7dd1a272f01700b13d6cd7e4b9bc704",
      "age": 13,
      "zcredits": 0
    },
    {
      "coin": "BTC",
      "address": "1M7RX4n5qtJLHZx2Wht2ZHwcWvpzstzcRW",
      "price": 0.00033395,
      "numutxos": 1,
      "avevolume": 0,
      "maxvolume": 0,
      "depth": 0.20000000,
      "pubkey":
"e1d9ff940bd7fb30954e0d9297053457eb7f766e66da6c198e75c06b7d982124",
      "age": 4,
      "zcredits": 0
    },
    {
```

For enabling coins when you have a native coin daemon running, the `enable` request is needed. Copy the following curl command in a new Insomnia request, called `enable <coin>`. I use ZEC in this example.

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"ZEC\"}"
```

This enables ZEC to be used in marketmaker for the current session. If you stop marketmaker and start it again, you need to enable ZEC again before it can be used in marketmaker. This goes for every other coin, except KMD and BTC.

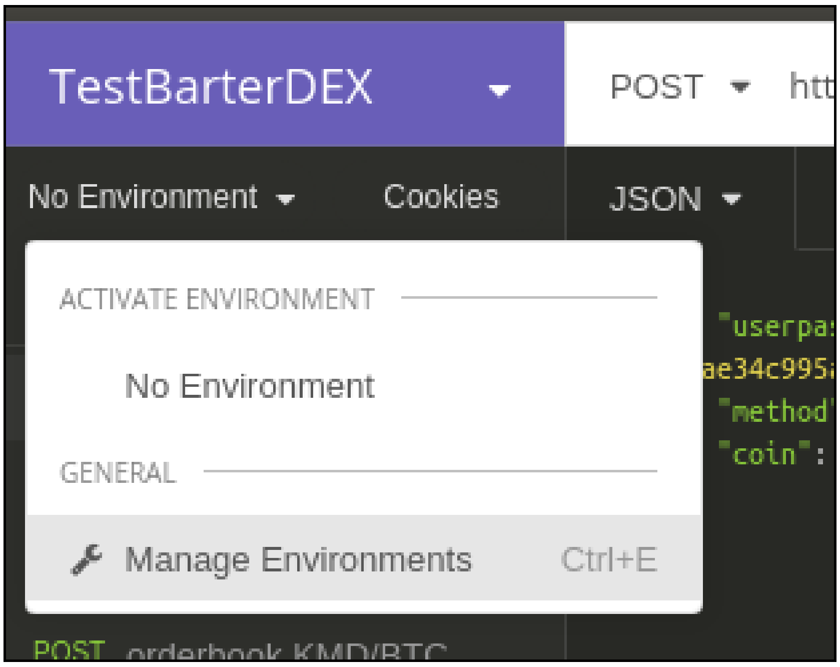
To avoid having to enable coins everytime you start marketmaker, you need to edit the coins file: *How to edit the coins file*.

3.2.3 Environment variables

Now that we made a couple of requests, suppose you want to change the passphrase, and consequently the userpass. You'll have to go through every request and change the `userpass` value. With 25 requests, this becomes a bit tedious.

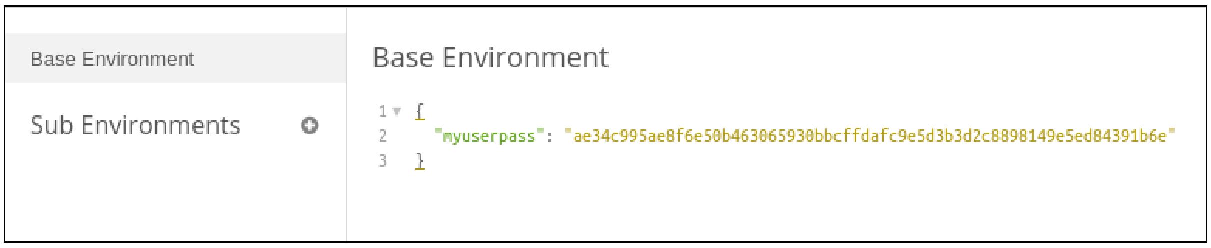
Using environment variables, we can set a variable and use it in each request. This way, you only need to change a value once, and it will apply to all the requests.

Click on No Environment and click Manage Environments (or press Ctrl-E):



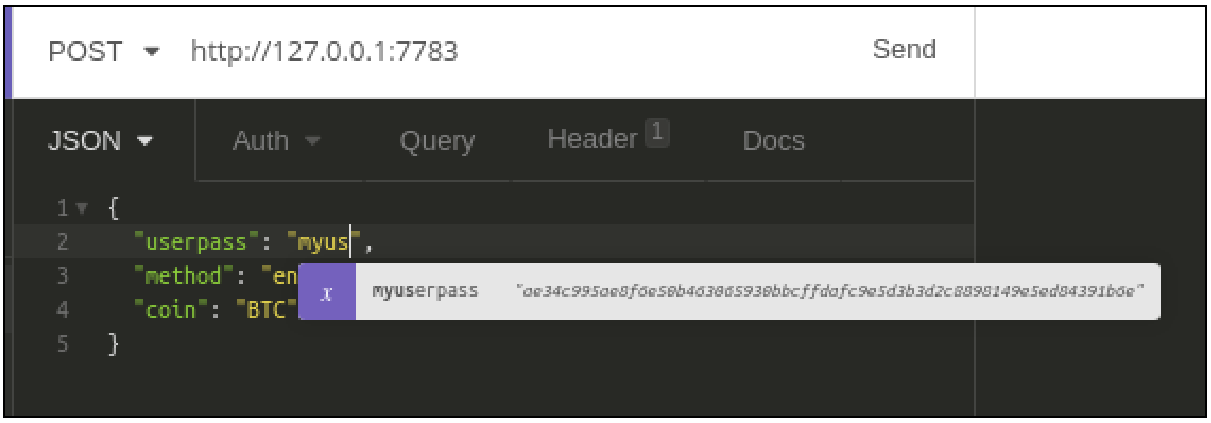
```
"userpa
ae34c995
"method"
"coin":
```

This leads you to the Base Environment, showing an empty JSON file. This JSON file will store all the global variables. Let's make a global `myuserpass` variable, containing your userpass.



```
{
  "myuserpass": "ae34c995ae8f6e50b463065930bbcffdafc9e5d3b3d2c8898149e5ed84391b6e"
}
```

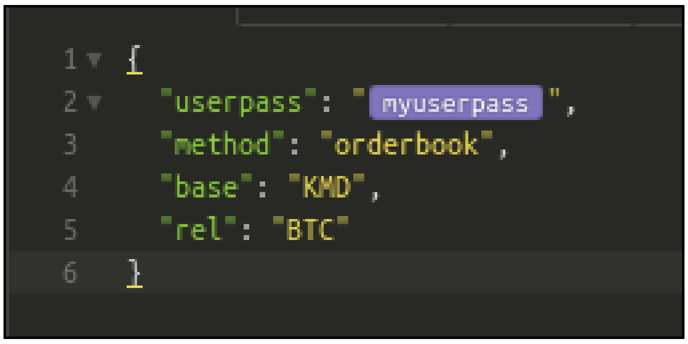
Click Done in the right bottom corner. Now, choose a request and remove the userpass value, but leave the 2 quotes, such that the following appears: `"userpass":  ""`. Start typing `myuserpass` in between the quotes, and wait for a dropdown to appear. When it appears, select the `myuserpass` value you see:



```
{
    "userpass": "myus",
    "method": "en
    "coin": "BTC"
}
```

```
x  myuserpass  "ae34c995ae8f6e50b463065930bbcffdafc9e5d3b3d2c8898149e5ed84391b6e"
```

If done correctly, you'll see this purple box appear between the 2 quotes. Do this for all your requests, except the `setpassphrase` request, since that requires a default passphrase for initial start-up.



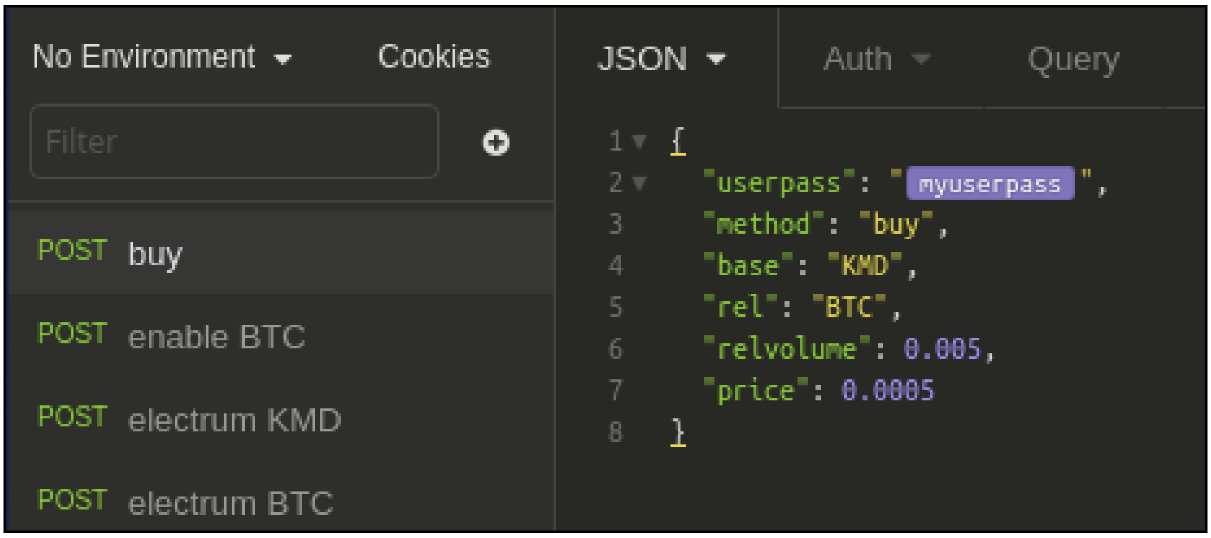
```
{
  "userpass": "myuserpass",
  "method": "orderbook",
  "base": "KMD",
  "rel": "BTC"
}
```

3.2.4 Buy request

We have successfully fetched an orderbook for the KMD/BTC pair. Let's try to buy something.

Copy the following `buy` curl and paste it in a new Insomnia request:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"{
↪{myuserpass}}\",\"method\":\"buy\",\"base\":\"KMD\",\"rel\":\"BTC\
↪",\"relvolume\":0.005,\"price\":0.0005}"
```



```
{
  "userpass": "myuserpass",
  "method": "buy",
  "base": "KMD",
  "rel": "BTC",
  "relvolume": 0.005,
  "price": 0.0005
}
```

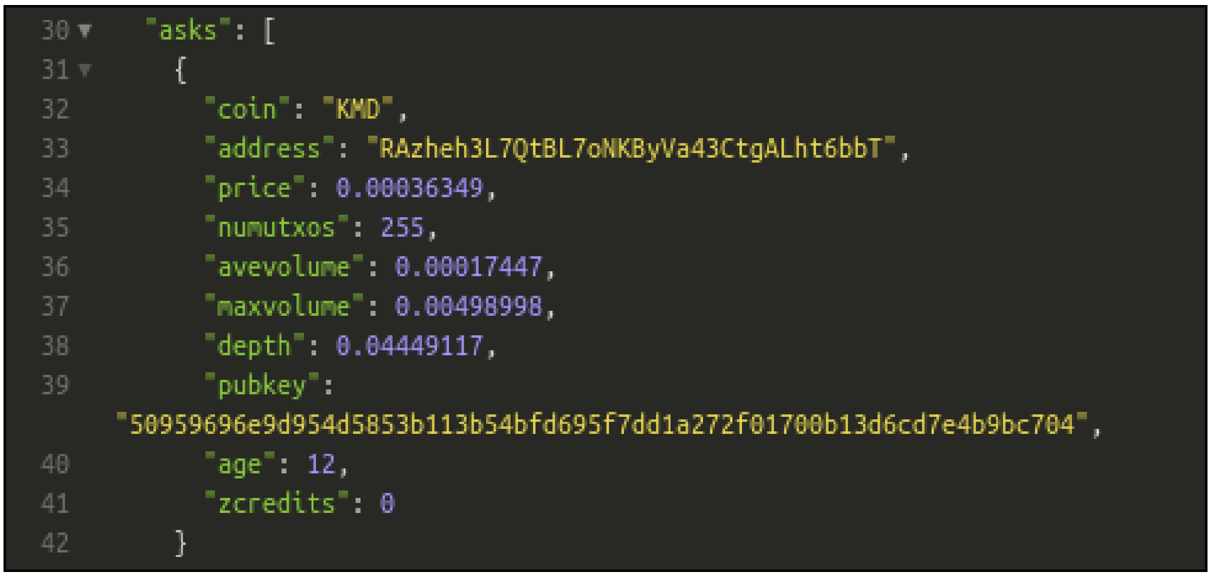
Buying a coin with another coin always happens in pairs: KMD/BTC means that KMD is the base coin and BTC is the rel coin: base/rel. You always buy base with rel, so in this case, you buy KMD with BTC. This means that you need some BTC in your BTC smart address, to be able to buy KMD with it, and you need to have at least 2 BTC transactions in this smart address, because you are paying a small *dexfee* too.

(For reasons why 2 transactions (UTXOs) are needed, read the *Overview of the atomic swap protocol*.)

You can get this BTC smart address by selecting the `setpassphrase` request and finding the `"BTC": "<address here>"` line.

`rel` and `relvolume` are related. The amount of KMD you can buy, depends on the `relvolume` you define. A glance at the orderbook will give you an idea about what size `relvolume` should be. Also, the amount of KMD you receive is a combination of relvolume and price. It is a result of a trade, rather than a goal. It is not simply saying: "I want 10 KMD, figure out how much BTC I need to pay". Instead, it is the other way around: "Here is 1 BTC, figure out how much KMD I get".

Take for example this order in the orderbook for KMD/BTC:



```
  "asks": [
    {
      "coin": "KMD",
      "address": "RAzheh3L7QtBL7oNKByVa43CtgALht6bbT",
      "price": 0.00036349,
      "numutxos": 255,
      "avevolume": 0.00017447,
      "maxvolume": 0.00498998,
      "depth": 0.04449117,
      "pubkey":
"50959696e9d954d5853b113b54bfd695f7dd1a272f01700b13d6cd7e4b9bc704",
      "age": 12,
      "zcredits": 0
    }
```

There are three things in this ask that you need to pay attention to:

- avevolume means the average volume of KMD this seller has to offer, expressed in BTC. This is useful, because you need to define the relvolume in your buy request as BTC.
- maxvolume means that the amount defined here is the largest KMD utxo of this seller.
- depth is a phrase commonly used by traders. Here it defines the sum of all KMD utxos from all sellers of KMD, cumulatively.

If you successfully want to buy this order, you need to adapt the relvolume in your buy request to the volume specified in this order. For this order, it would need to be lower than 0.00498998 BTC. It also depends on the utxo set of the seller, because a seller needs to be able to do a deposit and the actual payment. A deposit is about 13% larger than the actual payment, which means that the seller has to have 2 utxos of about the same size, in order to sell something of that size. To read more about this, read the *Overview of the atomic swap protocol*.

It is also important to state a price in your buy request. The price you define here is the maximum price you want to pay for your KMD. Since the order has set a price of 0.00036349, your max price needs to be a little above it, for your request to find willing counterparties to respond.

Now that we know all the information for submitting a buy request, we can create it. The following should, if the seller has enough utxos and if the atomic swap completes successfully, yield you some KMD (based on the data shown above):

```
{
   "userpass": "{{myuserpass}}",
   "method": "buy",
```

(continues on next page)

(continued from previous page)

```
    "base": "KMD",
    "rel": "BTC",
    "relvolume": 0.001,
    "price": 0.00037000
}
```

Note: Because of network propagation times and the use of cached data, the data shown in the orderbook is not always 100% up-to-date.

Contrary to centralised exchanges like Binance or Bittrex, submitting the `buy` request to the BarterDEX network by clicking *Send* in Insomnia does not fulfill this order automatically. Instead, what marketmaker does is sending a `buy request` onto the decentralised network, to all the other marketmaker nodes your node is connected to. Each node will see this request, and if there is a node that has a price set for this KMD/BTC pair, it will evaluate your request and if it fulfills all requirements (such as price and correct relvolume), it will reply with a message containing a counteroffer. This counteroffer is always better than the `buy request` you initially sent, because it falls within the boundaries you set in your `buy request`.

Your `buy request` can have multiple nodes on the marketmaker network respond to it. This all depends on the boundaries you set and the number of marketmaker nodes in the network. Your node then picks the best (lowest) price and starts the trade.

However! Defining a buy request that fulfills an order from the orderbook will not guarantee you a successful trade. Network issues could stop a trade, as well as out-of-date data from the seller. The orderbook may be stating that a seller has utxos, but in reality someone else could have bought the order already.

// TODO: hidden sellers (orders that exist but are not shown in orderbook) // TODO: switch pairs, switch price

3.2.5 Folders

3.2.6 Filtering

3.2.7 Electrum calls

3.2.8 History

Insomnia stores a list of all the calls you did in the past, including its output. This is useful for debugging and retrieving information you might need at a later stage.

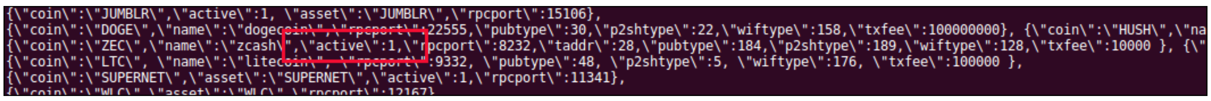
3.3 How to edit the coins file

The coins file contains all the currently supported coins in BarterDEX. It is loaded into marketmaker each time you start it. The file can be edited to your likings, mainly to enable coins on marketmaker startup automatically, without having to enable coins one by one.

Open the coins file with nano:

```
cd ~/SuperNET/iguana/dexscripts
nano coins
```

This will show you a list of all coins. Suppose you want to enable ZEC everytime you start marketmaker. You need to add the argument `\"active\":1,` to that JSON object, such that it results like this:



```
{\"coin\":\"JUMBLR\",\"active\":1, \"asset\":\"JUMBLR\",\"rpcport\":15106},
{\"coin\":\"DOGE\",\"name\":\"dogecoin\",\"rpcport\":22555,\"pubtype\":30,\"p2shtype\":22,\"wiftype\":158,\"txfee\":100000000}, {\"coin\":\"HUSH\",\"na
{\"coin\":\"ZEC\",\"name\":\"zcash\",\"active\":1,\"rpcport\":8232,\"taddr\":28,\"pubtype\":184,\"p2shtype\":189,\"wiftype\":128,\"txfee\":10000 }, {\"
{\"coin\":\"LTC\", \"name\":\"litecoin\", \"rpcport\":9332, \"pubtype\":48, \"p2shtype\":5, \"wiftype\":176, \"txfee\":100000 },
{\"coin\":\"SUPERNET\",\"asset\":\"SUPERNET\",\"active\":1,\"rpcport\":11341},
{\"coin\":\"WLC\",\"asset\":\"WLC\",\"rpcport\":12167}
```

To find a coin in this long list of supported coins, type Ctrl-W to search.

You can make as many coins active as you like. If the native daemon of an active coin is not running, marketmaker will ignore the coin and leave the coin disabled.

3.4 How to create a new BarterDEX trading network

Since BarterDEX is a decentralized, peer-to-peer network, seeded by some ip-addresses to create the network, others can create a BarterDEX network of their own. This enables people to trade within a private group of traders, or to trade directly from person to person.

See *New or private network* in the API docs on how to do this. (there currently is no GUI to handle this process)

CHAPTER 4

API docs

WORK IN PROGRESS! For a list of all API commands available in BarterDEX, refer to *API Summary by Category*. WORK IN PROGRESS!

4.1 Introduction

BarterDEX uses a custom-made peer-to-peer network and has Remote Procedure Calls (RPC) to talk with and get information from the BarterDEX network. If you installed BarterDEX by following the *How to use BarterDEX using CLI* guide, you'll have a lot of these commands ready to use in the `~/SuperNET/iguana/dexscripts` folder. These API docs will explain what each command does and what the possible arguments for each method are.

`marketmaker` is the system process that is started on a machine.

You need to set a strong passphrase (prior to starting marketmaker) and userpass to be able to talk to BarterDEX using RPC. The passphrase is what determines the addresses and the userpass value, and will be needed in the startup arguments when starting the marketmaker process on your machine (see below).

Curl can be used to send commands, see the following example taken from the `dexscripts` folder:

```
#!/bin/bash
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"autoprice\",\"base\":\"KMD\",\"rel\":\
↪"BTC\",\"margin\":0.0001}"
```

The userpass is sourced from the `dexscripts` folder, so that file has to be created in the `dexscripts` folder prior to sending a command. The userpass is there to prevent bad actors from issuing RPC commands. Marketmaker will not work without it set. Each command to BarterDEX will need this userpass value as value of the `userpass` key in the JSON.

`127.0.0.1:7783` is the ip address and port where BarterDEX is listening for commands at default, when no rpcport is defined in the startup arguments.

The json code in all the methods below is the data that is needed for each method described (mind the escape characters when using shell). For example:

```
{
     "userpass": "<userpass>",
     "method": "autoprice",
     "base": "KMD",
     "rel": "BTC",
     "margin": 0.0001
}
```

Replace everything that is shown as <this> with your own data.

4.2 Starting marketmaker

There are two different nodes in BarterDEX: a full relay node (Liquidity Provider, or LP, node) and a node that doesn't relay (client node). In the `dexscripts` folder, these are started by `./run` or `./client`, respectively. Normally, only Liquidity Providers will prefer a full relay node, as they can respond to incoming requests one hop sooner. Others will use `./client`, since it doesn't require as much bandwidth as a full relay node and can therefore be perfect for regular traders on BarterDEX.

4.2.1 Startup arguments

The following startup arguments need to be provided when starting the marketmaker binaries.

A full relay node (LP node) is started by running marketmaker with the following arguments string:

```
{
     "gui": "<name of gui>",
     "profitmargin": 0.01,
     "userhome": "<userhome> + /",
     "passphrase": "<passphrase>",
     "coins": ["<coins>"]
}
```

A node that doesn't relay (client node) has the following marketmaker startup arguments:

```
{
     "gui": "<name of gui>",
     "client": 1,
```

(continues on next page)

(continued from previous page)

```
    "userhome": "<userhome> + /",
    "passphrase": "<passphrase>",
    "coins": ["<coins>"]
}
```

- `gui` is the codename for the GUI used to start marketmaker with. If you are the developer of a GUI, you need to define a codename for your GUI. Share this in the Komodo Platform slack and you will get paid for every trade a user makes using your GUI.
- `profitmargin` is the default profitmargin that this LP node will use when placing orders in orderbooks using the `autoprice` method.
- `client`: when set to 1, it defines a client node.
- `userhome` is the location of the userhome.
- `passphrase` is the passphrase that is needed by `marketmaker` to determine the userpass and all smartaddresses that BarterDEX is going to use.
- `coins` needs a JSON of all BarterDEX-enabled coins. Not all cryptocurrencies are able to do atomic swaps, because they lack CheckLockTimeVerify (BIP65) or one of the necessary Bitcoin API methods (See *How to get listed on BarterDEX?* for details).

Optional:

- `wif` when set to 1, the `setpassphrase` API call will show WIF keys for all smartaddresses.

After `marketmaker` started successfully, the first RPC to be issued will always return a `getcoin` <REF TO GETCOIN> call for all coins, using 'default' as the default passphrase. This will also return the default userpass, which will need to be used to set the passphrase of the user, using the `passphrase` api call:

```
{
    "userpass":
↪"1d8b27b21efabcd96571cd56f91a40fb9aa4cc623d273c63bf9223dc6f8cd81f
↪",
    "method": "passphrase",
    "passphrase": "<passphrase>",
    "gui": "<name of gui>",
    "netid": 0
}
```

The `netid` needs to be defined when using a `netid` other than 0.

This method will return a response containing the `userpass` value for the user passphrase as defined in the `passphrase` method.

4.2.2 New or private network

In order to start a network other than the default network, you need to add at least 2 arguments to the `marketmaker` startup arguments. When initiating a new network, a full relay node must be used, and it has to define `"netid":<int netid>` and `"seednode":"<ipaddress>"` to the marketmaker startup arguments, where the netid is any integer higher than 0 but lower than 14420. The seednode is the ip address of the server being a full relay node.

Non-relay nodes (`client`) need to use the same 2 arguments in its startup arguments, to be able to join that network.

At default, the RPC port for a marketmaker instance is 7783. To override this setting, add `"rpcport":<int port>` to the startup arguments. This port can be any port in the range 1025 - 65535. Defining the RPC port is for local networking; other nodes in the network do not have to comply by having the same RPC port settings.

4.2.3 Multiple marketmaker instances

Multiple instances of marketmaker on the same machine are possible, by defining a different `netid`, `seednode` (optional) and `rpcport`. For example: One node is joining an existing network using `netid:0` and `rpcport:8800`. A second instance of marketmaker can now be started with `netid:1` and `rpcport:8810`. Each node has now access to a different network, and thus a different orderbook.

When initiating a new network, apart from defining the `netid`, the `seednode` has to be defined too. As long as the combination of `netid` and `seednode` does not exist yet, a new network will be created. Therefore, multiple networks can exist with `netid:0`, each with a different orderbook. The `seednode` is essential for defining multiple networks using the same `netid`. When no `seednode` is defined, the default seednodes are used, which essentially are the seednodes of the main BarterDEX network. No new network will then be created; the `marketmaker` instance will be joining the existing, main BarterDEX network.

This basically means that an almost infinite number of BarterDEX networks can be created, using the `netid` and `seednode` startup arguments for `marketmaker`.

4.3 General commands

4.3.1 orderbook

One of the most important calls in an exchange: getting to see the orderbook for a specific pair.

```
{
    "userpass": "<userpass>",
    "method": "orderbook",
    "base": "<base_coin>",
```

(continues on next page)

(continued from previous page)

```
    "rel": "<rel_coin>"
}
```

Output:

```
{
    "bids": [
        {
            "coin": "<rel_coin>",
            "address": "RKdCvGQZbjUf51ae6xsNu5by8tZL5ztjhW",
            "price": 0.11011000,
            "numutxos": 0,
            "avevolume": 0,
            "maxvolume": 0,
            "depth": 0,
            "pubkey":
↪"89274a7a0e93b850edb34907250ce9e3d3217b3d864326d0553bf3592a535c05
↪",
            "age": 55,
            "zcredits": 0
        }
    ],
    "numbids": 1,
    "biddepth": 0,
    "asks": [
        {
            "coin": "<base_coin>",
            "address": "RK5xVwfd1Qf8iuTymMUUri22rYxDW3396R",
            "price": 0.10000000,
            "numutxos": 4,
            "avevolume": 2.23920003,
            "maxvolume": 2.40000003,
            "depth": 8.95680013,
            "pubkey":
↪"198a41d6259ab7585d7dd566966375d21361d191d59c698bf3d6e9f47df99f7c
↪",
            "age": 20,
            "zcredits": 0
        }
    ],
    "numasks": 1,
    "askdepth": 11.19600016,
    "base": "<base_coin>",
    "rel": "<rel_coin>",
    "timestamp": 1520187231,
    "netid": 0
}
```

Optional:

- fetching orderbook
- get coin info, smart addy etc
- balance(s)
- listunspent
- swapstatus

4.4 Price commands

Most, if not all, of the trade commands use the base/rel notation of pricing orders.

4.4.1 autoprice

The autoprice command is a rich command that allows anyone to create an order using data from CoinMarketCap or any other exchange. It refreshes the price every 1-2 minutes, such that once the autoprice command is executed, the order will be in the orderbooks permanently.

There are several possibilities for autoprice:

fixed price

The following command puts an ask in the BTC/KMD orderbook and basically says: ‘I want to get KMD by selling BTC at a fixed price of 1800’. So, anyone who wants to buy BTC with KMD will see this order and can buy 1 BTC for 1800 KMD.

```
{
     "userpass": "<userpass>",
     "method": "autoprice",
     "base": "KMD",
     "rel": "BTC",
     "fixed": 1800
}
```

price with margin

<NEED TO ASK WHAT THIS DOES EXACTLY>

```
{
     "userpass": "<userpass>",
     "method": "autoprice",
     "base": "KMD",
     "rel": "BTC",
     "margin": 0.01
}
```

price based on external data

The following command would refresh the price of the order in the orderbook based on price changes as defined in the `refrel` argument:

<NEED MORE INFO>

```
{
      "userpass": "<userpass>",
      "method": "autoprice",
      "base": "KMD",
      "rel": "BTC",
      "margin": 0.05,
      "refbase": "kmd",
      "refrel": "coinmarketcap"
}
```

Note: the base and rel need to be uppercase and the refbase needs to be lowercase

4.5 UTXO tools

4.5.1 withdraw

4.5.2 sendrawtransaction

4.6 Address tools

4.6.1 calcaddress

Returns the address, wif and public key for the passphrase defined.

```
{
      "userpass": "<userpass>",
      "method": "calcaddress",
      "passphrase": "<passphrase>"
}
```

Output (for passphrase `default`):

```
{
      "passphrase": "default",
      "coinaddr": "RPZVpjptzfZnFZZoLnuSbfLexjtkhe6uvn",
      "privkey":
↪"30a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0684f
↪",
```

(continues on next page)

(continued from previous page)

```
    "wif": "Uqe8cy26KvC2xqfh3aCpKvKjtoLC5YXiDW3iYf4MGSSy1RgMm3V5"
}
```

4.7 Docker

lukechilds has provided a docker image for BarterDEX.

CHAPTER 5

API Summary by Category

5.1 List of API by Category

Click any of the api options below to be taken to their summary. Then navigate using the links in the Navigation bar on the left side. Or press the `Home` button on your keyboard to come back to the top again. As the docs are full of code samples and terminal outputs, it's best viewed on a laptop/desktop (i.e, any device that is bigger than a handheld).

5.1.1 barterDEX Operation

client, *client_osx*, *convaddress*, *debug*, *disable*, *electrum*, *enable*, *getcoin*, *getcoins*, *getpeers*, *getpeersIP*, *help*, *jpg*, *millis*, *notarizations*, *parselog*, *run*, *setpassphrase*, *sleep*, *stop*, *trust*, *trusted*

5.1.2 barterDEX Trading

autoprice, *buy*, *getfee*, *getprices*, *goal*, *goals*, *myprice*, *myprices*, *orderbook*, *pubkeystats*, *sell*, *setconfirms*, *setprice*, *fomo*, *dump*

> *autoprice the value of a fund, ie. using the fundvalue api*

> *autoprice using usdpeg*

5.1.3 Status / Info

getendpoint, *pendings*, *swapstatus*, *baserelswaps*, *pendingswaps*, *coinswaps*, *swapstatus(requestid, quoteid, pending=0)*, *recentswaps*

5.1.4 TradeBots

bot_buy, *bot_list*, *bot_pause*, *bot_resume*, *bot_sell*, *bot_settings*, *bot_status*, *bot_stop*

5.1.5 Coin Wallet Features

balance, *balances*, *calcaddress*, *fundvalue*, *getrawtransaction*, *inuse*, *listtransactions*, *listunspent*, *secretaddresses*, *sendrawtransaction*, *supernet*, *timelock and unlockedspend*, *withdraw*, *eth_withdraw*, *opreturn*, *opreturndecrypt*

5.1.6 Statistics

guistats, *pricearray*, *statsdisp*, *ticker*, *tradesarray*

5.1.7 Communication

deletemessages, *getmessages*, *message*

5.1.8 Revenue Sharing/Operations

dividends, *snapshot*, *snapshot_balance*, *snapshot_loop*

InstantDEX SWAP - `deposit` & `claim`

5.1.9 Docker

Use command line JSON args, either `"docker":1` or `"docker":"<ipaddr>"` with marketmaker and the request must come from that ip address to be mapped to localhost.

If you have docker installed you can get the BarterDEX API running on Windows, Mac or Linux by running:

```
docker run -e PASSPHRASE="secure passphrase" -p 127.0.0.1:7783:7783␣
↪lukechilds/barterdex-api
```

GitHub link

Summary info for each API by Category:

5.2 BarterDEX Operation

5.2.1 client

The first API to run which will start barterDEX in client mode. Next script to run is `setpassphrase`. If you want to close barterDEX, issue `pkill -15 marketmaker` every time. This ensures all barterDEX process is killed safely.

Sample File Contents:

```
#!/bin/bash
source passphrase
source coins
./stop
git pull;
cp ../exchanges/updateprices .;./updateprices
cd ..;
./m_mm;
pkill -15 marketmaker;
stdbuf -oL $1 ./marketmaker "{\"gui\":\"nogui\",\"client\":1,\
↪"userhome\":\"/${HOME#"/"}\",\"passphrase\":\"$passphrase\",\
↪"coins\":$coins}" &
```

Fields that can be used: `gui:nogui` or `gui:SimpleUI` - to identify the CLI/GUI being used to run marketmaker. `client:1` - to start marketmaker as client mode. `"userhome\":\"/${HOME#"/"` - defines the data dir for barterDEX. DB dir contains PRICES, SWAPS, UNSPENTS and instantdex_deposit (0conf) files. `passphrase:$passphrase` - is to set the passphrase to run the marketmaker with, to run and login into marketmaker using WIF (Wallet Import Format) key instead of passphrase. You need to use the WIF key as passphrase inside `passphrase` file. barterDEX supports login using WIF Key for all supported coins instead of passphrase. `wif:1` - will display the passphrase's WIF during the first api return. `rpcport:port` - to change the 7782 port to other open port of choice.

5.2.2 netid

For the startup JSON default is 0, max is 19240, which should be plenty of p2p networks for a while. Each `netid` will operate totally independent of the other netids. The `rpcport` field can be used to specify different ones so you can run multiple mm on the same node, but they will be on separate `netid`. This way you can broadcast the `marketmaker` prices to all the netids. You also need to use the same `netid` in `setpassphrase` script.

Example usage:

```
./marketmaker "{\"gui\":\"nogui\",\"client\":1, \"userhome\":\"/$
↪{HOME#"/"}\", \"passphrase\":\"$passphrase\", \"coins\":$coins, \
↪"netid\":1}"
```

5.2.3 client_osx

It is the same script as above but only for MacOS.

Sample File Contents:

```
#!/bin/bash
source passphrase
source coins
pkill -15 marketmaker;
git pull;
cd ..;
./m_mm;
./marketmaker "{\"gui\":\"nogui\",\"client\":1, \"userhome\":\"/$
↪{HOME#"/"}/Library/Application\ Support\", \"passphrase\":\"
↪$passphrase\", \"coins\":$coins}" &
```

5.2.4 convaddress

This script should work with p2sh addresses and also with normal addresses to convert. You just have to specify the `coin`, `address` & `destcoin` parameter and run.

Sample file contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"convaddress\",\"coin\":\"BTC\",\"address\
↪":\"1KPctPk4Zs4Qbe1x32A5bC1roAnmpvi9Fy\",\"destcoin\":\"KMD\"}"
```

Sample Output 1:

```
{
    "result": "success",
    "coin": "BTC",
    "address": "1KPctPk4Zs4Qbe1x32A5bC1roAnmpvi9Fy",
    "destcoin": "KMD",
    "destaddress": "RTfoxudMAgryfeP9WC9CgiM4ZSFNVM2VvK"
}
```

Sample Output 2:

```
{
    "result": "success",
    "coin": "BTC",
    "address": "1KPctPk4Zs4Qbe1x32A5bC1roAnmpvi9Fy",
    "destcoin": "DGB",
    "destaddress": "DPXiReghsGxh8eCYmc9e8xBTgJX5CdnALu"
}
```

5.2.5 debug

This script will let you debug if you run into problem with `marketmaker` and coins using `gdb`. This will kill the `marketmaker` first, do a `git pull` for fresh data and build it again. Once ready, type `run` and hit enter to start. In case of marketmaker crash, you can use `backtrace` command to find the reason of the crash. To exit from this mode type `quit` and hit enter.

Sample File Contents:

```
#!/bin/bash
source passphrase
source coins
pkill -15 marketmaker;
git pull;
cd ..;
./m_mm;
gdb -args ./marketmaker "{\"gui\":\"nogui\",\"client\":1,\"userhome\
↪":\"/${HOME#"/"}\",\"passphrase\":\"$passphrase\",\"coins\":
↪$coins}"
```

5.2.6 disable

This method disables a coin. A disabled coin is not eligible for trading.

Sample File Contents:

```
#!/bin/bash
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":       \"
↪$userpass\",\"method\":\"disable\",\"coin\":\"REVS\"}"
```

5.2.7 enable

This method enables a coin by connecting to a locally running native coin daemon, otherwise, known as **Native Mode**. To be eligible for trading a coin must be enabled and the daemon should be running. You can edit the sample `enable` script file to activate the native coins you want to trade. you can stop it by using the *disable* script.

Sample File Contents:

```
#!/bin/bash
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"BEER\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"ETOMIC\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"PIZZA\"}"
```

(continues on next page)

(continued from previous page)

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"REVS\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"KMD\"}"
#curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"BTC\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"CHIPS\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"SUPERNET\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"CRYPTO\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"DEX\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"BOTS\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"BET\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"HODL\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"MSHARK\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"MGW\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"PANGEA\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"JUMBLR\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"HUSH\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"enable\",\"coin\":\"BTCH\"}"
```

Sample Output:

```
[{
    "coin": "KMD",
    "installed": true,
    "height": 706875,
    "balance": 1231.55116115,
    "KMDvalue": 1231.55116115,
    "status": "active",
    "electrum": "electrum1.cipig.net:10001",
    "smartaddress": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
    "rpc": "127.0.0.1:7771",
    "pubtype": 60,
    "p2shtype": 85,
    "wiftype": 188,
    "txfee": 1000,
```

(continues on next page)

(continued from previous page)

```
    "zcredits": 1000,
    "zdebits": {
        "swaps": [{
            "iambob": 1,
            "aliceid": 14646443851645911040,
            "requestid": 1531405035,
            "quoteid": 668442874,
            "base": "KMD",
            "satoshis": 398119479,
            "rel": "BTCH",
            "destsatoshis": 996001000,
            "price": 2.50176404,
            "finished": 0,
            "bobneeds_dPoW": 1,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 1,
            "alice_dPoWheight": 0,
            "kmdvalue": 3.98119479
        }, {
            "iambob": 0,
            "aliceid": 7500808088360189952,
            "requestid": 2426472678,
            "quoteid": 3136038708,
            "base": "KMD",
            "satoshis": 170727131,
            "rel": "MSHARK",
            "destsatoshis": 30001000,
            "price": 0.17572485,
            "finished": 1518455358,
            "bobneeds_dPoW": 706621,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 20948,
            "alice_dPoWheight": 14108,
            "kmdvalue": 1.72646183
        }, {
            "iambob": 0,
            "aliceid": 10968666690691268608,
            "requestid": 769582120,
            "quoteid": 492899755,
            "base": "KMD",
            "satoshis": 57319224,
            "rel": "MSHARK",
            "destsatoshis": 10001000,
            "price": 0.17447898,
            "finished": 1518455358,
            "bobneeds_dPoW": 706644,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 20963,
            "alice_dPoWheight": 14108,
```

(continues on next page)

(continued from previous page)

```
            "kmdvalue": 0.57552564
        }, {
            "iambob": 0,
            "aliceid": 3325348207487614976,
            "requestid": 1461677939,
            "quoteid": 3994513936,
            "base": "KMD",
            "satoshis": 57334419,
            "rel": "MSHARK",
            "destsatoshis": 10001000,
            "price": 0.17443274,
            "finished": 1518455358,
            "bobneeds_dPoW": 706648,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 20965,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.57552564
        }, {
            "iambob": 0,
            "aliceid": 3949043092670119936,
            "requestid": 868202615,
            "quoteid": 2718753539,
            "base": "KMD",
            "satoshis": 40174013,
            "rel": "MSHARK",
            "destsatoshis": 7001000,
            "price": 0.17426688,
            "finished": 1518455358,
            "bobneeds_dPoW": 706662,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 20972,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.40288521
        }, {
            "iambob": 0,
            "aliceid": 1187687474903711744,
            "requestid": 1035974224,
            "quoteid": 4214136299,
            "base": "KMD",
            "satoshis": 40170553,
            "rel": "MSHARK",
            "destsatoshis": 7001000,
            "price": 0.17428189,
            "finished": 1518455358,
            "bobneeds_dPoW": 706665,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 20975,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.40288521
```

(continues on next page)

(continued from previous page)

```
        }, {
            "iambob": 0,
            "aliceid": 3361541373681205248,
            "requestid": 1030751685,
            "quoteid": 1769312982,
            "base": "KMD",
            "satoshis": 40197682,
            "rel": "MSHARK",
            "destsatoshis": 7001000,
            "price": 0.17416427,
            "finished": 1518455358,
            "bobneeds_dPoW": 706665,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 20982,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.40288521
        }, {
            "iambob": 0,
            "aliceid": 16301556677736726528,
            "requestid": 1082032834,
            "quoteid": 977860445,
            "base": "KMD",
            "satoshis": 40191021,
            "rel": "MSHARK",
            "destsatoshis": 7001000,
            "price": 0.17419313,
            "finished": 1518455358,
            "bobneeds_dPoW": 706669,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 20991,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.40288521
        }, {
            "iambob": 0,
            "aliceid": 12486716361556295680,
            "requestid": 131073424,
            "quoteid": 554811147,
            "base": "KMD",
            "satoshis": 57472918,
            "rel": "MSHARK",
            "destsatoshis": 10001000,
            "price": 0.17401239,
            "finished": 1518455358,
            "bobneeds_dPoW": 706718,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 21033,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.57552564
        }, {
```

(continues on next page)

(continued from previous page)

```
            "iambob": 0,
            "aliceid": 13831903913886154752,
            "requestid": 3159266235,
            "quoteid": 2688690321,
            "base": "KMD",
            "satoshis": 57143474,
            "rel": "MSHARK",
            "destsatoshis": 10001000,
            "price": 0.17501561,
            "finished": 1518455358,
            "bobneeds_dPoW": 706715,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 21029,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.57552564
        }, {
            "iambob": 0,
            "aliceid": 10301729584199958528,
            "requestid": 1213522022,
            "quoteid": 54249754,
            "base": "KMD",
            "satoshis": 57528414,
            "rel": "MSHARK",
            "destsatoshis": 10001000,
            "price": 0.17384453,
            "finished": 1518455358,
            "bobneeds_dPoW": 706719,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 21038,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.57552564
        }, {
            "iambob": 0,
            "aliceid": 8232914266839056384,
            "requestid": 4081601584,
            "quoteid": 412356287,
            "base": "KMD",
            "satoshis": 57494778,
            "rel": "MSHARK",
            "destsatoshis": 10001000,
            "price": 0.17394623,
            "finished": 0,
            "bobneeds_dPoW": 1,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 1,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.57552564
        }, {
            "iambob": 0,
```

(continues on next page)

(continued from previous page)

```
            "aliceid": 9209632436025032704,
            "requestid": 3537842224,
            "quoteid": 3725087089,
            "base": "KMD",
            "satoshis": 21597791,
            "rel": "MSHARK",
            "destsatoshis": 3747597,
            "price": 0.17351760,
            "finished": 1518455358,
            "bobneeds_dPoW": 706758,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 21082,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.21566225
        }, {
            "iambob": 0,
            "aliceid": 8233074053957746688,
            "requestid": 2261166303,
            "quoteid": 22701758,
            "base": "KMD",
            "satoshis": 21597177,
            "rel": "MSHARK",
            "destsatoshis": 3746456,
            "price": 0.17346970,
            "finished": 1518455358,
            "bobneeds_dPoW": 706765,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 21091,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.21559659
        }, {
            "iambob": 0,
            "aliceid": 6122187997339189248,
            "requestid": 1197866694,
            "quoteid": 488405108,
            "base": "KMD",
            "satoshis": 115069202,
            "rel": "MSHARK",
            "destsatoshis": 20001000,
            "price": 0.17381714,
            "finished": 1518455358,
            "bobneeds_dPoW": 706766,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 21093,
            "alice_dPoWheight": 14108,
            "kmdvalue": 1.15099373
        }, {
            "iambob": 0,
            "aliceid": 14721798049184415744,
```

(continues on next page)

(continued from previous page)

```
            "requestid": 739807218,
            "quoteid": 364988835,
            "base": "KMD",
            "satoshis": 115164687,
            "rel": "MSHARK",
            "destsatoshis": 20001000,
            "price": 0.17367303,
            "finished": 1518455358,
            "bobneeds_dPoW": 706773,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 21101,
            "alice_dPoWheight": 14108,
            "kmdvalue": 1.15099373
        }, {
            "iambob": 0,
            "aliceid": 15764304953332531200,
            "requestid": 2360598697,
            "quoteid": 3777086736,
            "base": "KMD",
            "satoshis": 57519187,
            "rel": "MSHARK",
            "destsatoshis": 10001000,
            "price": 0.17387241,
            "finished": 0,
            "bobneeds_dPoW": 1,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 1,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.57552564
        }, {
            "iambob": 0,
            "aliceid": 15764470458444021760,
            "requestid": 1183002364,
            "quoteid": 2238731236,
            "base": "KMD",
            "satoshis": 20796927,
            "rel": "MSHARK",
            "destsatoshis": 3600230,
            "price": 0.17311355,
            "finished": 1518455358,
            "bobneeds_dPoW": 706822,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 21145,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.20718175
        }, {
            "iambob": 0,
            "aliceid": 15223997733381865472,
            "requestid": 1894968837,
```

(continues on next page)

(continued from previous page)

```
            "quoteid": 948074537,
            "base": "KMD",
            "satoshis": 20806039,
            "rel": "MSHARK",
            "destsatoshis": 3594169,
            "price": 0.17274642,
            "finished": 1518456523,
            "bobneeds_dPoW": 706844,
            "bob_dPoWheight": 687380,
            "aliceneeds_dPoW": 21189,
            "alice_dPoWheight": 14108,
            "kmdvalue": 0.20683295
        }],
        "pendingswaps": 14.49513794
    }
}]
[{
    "coin": "SUPERNET",
    "installed": true,
    "height": 85017,
    "balance": 0.94386703,
    "KMDvalue": 35.77755890,
    "status": "active",
    "smartaddress": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
    "rpc": "127.0.0.1:11341",
    "pubtype": 60,
    "p2shtype": 85,
    "wiftype": 188,
    "txfee": 1000
}]
[{
    "coin": "BOTS",
    "installed": true,
    "height": 14118,
    "balance": 8.61219046,
    "KMDvalue": 95.90388479,
    "status": "active",
    "electrum": "electrum1.cipig.net:10007",
    "smartaddress": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
    "rpc": "127.0.0.1:11964",
    "pubtype": 60,
    "p2shtype": 85,
    "wiftype": 188,
    "txfee": 1000
}]
[{
    "coin": "BET",
    "installed": true,
    "height": 16555,
```

(continues on next page)

(continued from previous page)

```
    "balance": 36.65094825,
    "KMDvalue": 0,
    "status": "active",
    "electrum": "electrum1.cipig.net:10012",
    "smartaddress": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
    "rpc": "127.0.0.1:14250",
    "pubtype": 60,
    "p2shtype": 85,
    "wiftype": 188,
    "txfee": 1000
}]
[{
    "coin": "MSHARK",
    "installed": true,
    "height": 21221,
    "balance": 61.90419488,
    "KMDvalue": 356.23889156,
    "status": "active",
    "smartaddress": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
    "rpc": "127.0.0.1:8846",
    "pubtype": 60,
    "p2shtype": 85,
    "wiftype": 188,
    "txfee": 1000
}]
[{
    "coin": "BTCH",
    "installed": true,
    "height": 17922,
    "balance": 1890.26497637,
    "KMDvalue": 1085.25152617,
    "status": "active",
    "smartaddress": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
    "rpc": "127.0.0.1:8800",
    "pubtype": 60,
    "p2shtype": 85,
    "wiftype": 188,
    "txfee": 1000
}]
```

5.2.8 getcoin

This method will show coin data including smartaddress, balance etc. Do NOT use `getcoin` to get balance in SPV mode, use the `balance` API.

Sample File Contents:

```
#!/bin/bash
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"getcoin\",\"coin\":\"LTC\"}"
```

Sample Output:

```
{
    "result": "success",
    "enabled": 5,
    "disabled": 88,
    "coin": {
        "coin": "BTCH",
        "installed": true,
        "height": 17914,
        "balance": 1890.26497637,
        "KMDvalue": 1060.52748251,
        "status": "active",
        "smartaddress": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
        "rpc": "127.0.0.1:8800",
        "pubtype": 60,
        "p2shtype": 85,
        "wiftype": 188,
        "txfee": 1000
    }
}
```

5.2.9 getcoins

This will display the list of all coins that barterDEX supports. It will list both disabled and enabled coins. Along with the status of the coin, this function will also display your smartaddress for that given coin. This method does not need user defined inputs and will just display all the coins.

Sample File Contents:

```
#!/bin/bash
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"getcoins\"}"
```

Sample Output:

```
[{
        "coin": "BTC",
        "height": 0,
        "status": "inactive",
        "smartaddress": "15XAhHK4ULofD6BTckrCWK6XSkCjscX7jj",
        "rpc": "127.0.0.1:8332",
```

(continues on next page)

(continued from previous page)

```
        "pubtype": 0,
        "p2shtype": 5,
        "wiftype": 128,
        "txfee": 0
    },
    {
        "coin": "KMD",
        "height": 538355,
        "status": "active",
        "smartaddress": "RDoMmoCM5AcEH6Yf5vqKbqRjD1fLcQHuWb",
        "rpc": "127.0.0.1:7771",
        "pubtype": 60,
        "p2shtype": 85,
        "wiftype": 188,
        "txfee": 10000
    }
]
```

5.2.10 getpeers

The `./getpeers` script prints the connected nodes available to trade in the network.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"getpeers\"}"
```

Sample Output:

```
[{
        "ipaddr": "5.9.253.202",
        "port": 7783,
        "session": 1508151814
    },
    {
        "ipaddr": "5.9.253.195",
        "port": 7783,
        "session": 1508151804
    },
    {
        "ipaddr": "5.9.253.196",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.197",
        "port": 7783,
        "session": 0
```

(continues on next page)

(continued from previous page)

```
    },
    {
        "ipaddr": "5.9.253.198",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.199",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.200",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.201",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.203",
        "port": 7783,
        "session": 0
    }
]
```

5.2.11 getpeersIP

The `./getpeersIP` script prints all the connected nodes, their `port` and `session` number.

Sample File Contents:

```
curl --url "http://5.9.253.195:7783" --data "{\"method\":\"getpeers\
↪"}"
curl --url "http://5.9.253.196:7783" --data "{\"method\":\"getpeers\
↪"}"
curl --url "http://5.9.253.197:7783" --data "{\"method\":\"getpeers\
↪"}"
curl --url "http://5.9.253.198:7783" --data "{\"method\":\"getpeers\
↪"}"
curl --url "http://5.9.253.199:7783" --data "{\"method\":\"getpeers\
↪"}"
curl --url "http://5.9.253.200:7783" --data "{\"method\":\"getpeers\
↪"}"
curl --url "http://5.9.253.201:7783" --data "{\"method\":\"getpeers\
↪"}"
```

(continues on next page)

(continued from previous page)

```
curl --url "http://5.9.253.202:7783" --data "{\"method\":\"getpeers\
↪"}"
curl --url "http://5.9.253.203:7783" --data "{\"method\":\"getpeers\
↪"}"
curl --url "http://5.9.253.204:7783" --data "{\"method\":\"getpeers\
↪"}"
```

Sample Output:

```
[{
        "ipaddr": "5.9.253.195",
        "port": 7783,
        "session": 1508151804
    },
    {
        "ipaddr": "5.9.253.196",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.197",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.198",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.199",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.200",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.201",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.203",
        "port": 7783,
        "session": 0
    },
```

(continues on next page)

(continued from previous page)

```
    {
        "ipaddr": "5.9.253.202",
        "port": 7783,
        "session": 0
    }
]
[{
        "ipaddr": "5.9.253.196",
        "port": 7783,
        "session": 1508151807
    },
    {
        "ipaddr": "5.9.253.195",
        "port": 7783,
        "session": 1508151804
    },
    {
        "ipaddr": "5.9.253.197",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.198",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.199",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.200",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.201",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.203",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.202",
        "port": 7783,
```

(continues on next page)

(continued from previous page)

```
        "session": 0
    }
]
[{
        "ipaddr": "5.9.253.197",
        "port": 7783,
        "session": 1508151808
    },
    {
        "ipaddr": "5.9.253.195",
        "port": 7783,
        "session": 1508151804
    },
    {
        "ipaddr": "5.9.253.196",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.198",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.199",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.200",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.201",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.203",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.202",
        "port": 7783,
        "session": 0
    }
]
```

(continues on next page)

(continued from previous page)

```
[{
        "ipaddr": "5.9.253.198",
        "port": 7783,
        "session": 1508151809
    },
    {
        "ipaddr": "5.9.253.195",
        "port": 7783,
        "session": 1508151804
    },
    {
        "ipaddr": "5.9.253.196",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.197",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.199",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.200",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.201",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.203",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.202",
        "port": 7783,
        "session": 0
    }
]
[{
        "ipaddr": "5.9.253.199",
        "port": 7783,
```

(continues on next page)

(continued from previous page)

```
            "session": 1508151809
        },
        {
            "ipaddr": "5.9.253.195",
            "port": 7783,
            "session": 1508151804
        },
        {
            "ipaddr": "5.9.253.196",
            "port": 7783,
            "session": 0
        },
        {
            "ipaddr": "5.9.253.197",
            "port": 7783,
            "session": 0
        },
        {
            "ipaddr": "5.9.253.198",
            "port": 7783,
            "session": 0
        },
        {
            "ipaddr": "5.9.253.200",
            "port": 7783,
            "session": 0
        },
        {
            "ipaddr": "5.9.253.201",
            "port": 7783,
            "session": 0
        },
        {
            "ipaddr": "5.9.253.203",
            "port": 7783,
            "session": 0
        },
        {
            "ipaddr": "5.9.253.202",
            "port": 7783,
            "session": 0
        }
]
[{
            "ipaddr": "5.9.253.200",
            "port": 7783,
            "session": 1508151812
        },
        {
```

(continues on next page)

(continued from previous page)

```
        "ipaddr": "5.9.253.195",
        "port": 7783,
        "session": 1508151804
    },
    {
        "ipaddr": "5.9.253.196",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.197",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.198",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.199",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.201",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.203",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.202",
        "port": 7783,
        "session": 0
    }
]
[{
        "ipaddr": "5.9.253.201",
        "port": 7783,
        "session": 1508151813
    },
    {
        "ipaddr": "5.9.253.195",
        "port": 7783,
        "session": 1508151804
```

(continues on next page)

(continued from previous page)

```
    },
    {
        "ipaddr": "5.9.253.196",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.197",
        "po###barterDEX Trading###rt": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.198",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.199",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.200",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.203",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.202",
        "port": 7783,
        "session": 0
    }
]
[{
        "ipaddr": "5.9.253.202",
        "port": 7783,
        "session": 1508151814
    },
    {
        "ipaddr": "5.9.253.195",
        "port": 7783,
        "session": 1508151804
    },
    {
        "ipaddr": "5.9.253.196",
```

(continues on next page)

(continued from previous page)

```
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.197",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.198",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.199",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.200",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.201",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.203",
        "port": 7783,
        "session": 0
    }
]
[{
        "ipaddr": "5.9.253.203",
        "port": 7783,
        "session": 1508151814
    },
    {
        "ipaddr": "5.9.253.195",
        "port": 7783,
        "session": 1508151804
    },
    {
        "ipaddr": "5.9.253.196",
        "port": 7783,
        "session": 0
    },
```

(continues on next page)

(continued from previous page)

```
    {
        "ipaddr": "5.9.253.197",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.198",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.199",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.200",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.201",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.202",
        "port": 7783,
        "session": 0
    }
]
[{
        "ipaddr": "5.9.253.197",
        "port": 7783,
        "session": 1506966021
    },
    {
        "ipaddr": "5.9.253.195",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.196",
        "port": 7783,
        "session": 0
    },
    {
        "ipaddr": "5.9.253.199",
        "port": 7783,
```

(continues on next page)

(continued from previous page)

```
            "session": 1506966023
        },
        {
            "ipaddr": "5.9.253.201",
            "port": 7783,
            "session": 1506966027
        },
        {
            "ipaddr": "5.9.253.202",
            "port": 7783,
            "session": 1506966028
        },
        {
            "ipaddr": "5.9.253.200",
            "port": 7783,
            "session": 0
        },
        {
            "ipaddr": "5.9.253.203",
            "port": 7783,
            "session": 0
        },
        {
            "ipaddr": "5.9.253.198",
            "port": 7783,
            "session": 0
        },
        {
            "ipaddr": "82.7.169.222",
            "port": 7783,
            "session": 0
        }
]
```

5.2.12 help

The `./help` API will display a list of all the available API available on barterDEX and usage options.

Sample File Contents:

```
#!/bin/bash
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"help\"}"
```

Sample Output:

```
{
"result": " available localhost RPC commands:
setprice(base, rel, price, broadcast = 1)
autoprice(base, rel, fixed, minprice, maxprice, margin, refbase,␣
↪refrel, factor, offset) *
goal(coin = * , val = < autocalc > )
myprice(base, rel)
enable(coin)
disable(coin)
notarizations(coin)
statsdisp(starttime = 0, endtime = 0, gui = , pubkey = , base = ,␣
↪rel = )
ticker(base = , rel = )
tradesarray(base, rel, starttime = < now > -timescale * 1024,␣
↪endtime = < now > , timescale = 60) - > [timestamp, high, low,␣
↪open, close, relvolume, basevolume, aveprice, numtrades]
pricearray(base, rel, starttime = 0, endtime = 0, timescale = 60) -␣
↪> [timestamp, avebid, aveask, highbid, lowask]
getrawtransaction(coin, txid)
inventory(coin, reset = 0, [passphrase = ])
lastnonce()
buy(base, rel, price, relvolume, timeout = 10, duration = 3600,␣
↪nonce)
sell(base, rel, price, basevolume, timeout = 10, duration = 3600,␣
↪nonce)
withdraw(coin, outputs[])
sendrawtransaction(coin, signedtx)
swapstatus(pending = 0, fast = 0)
swapstatus(coin, limit = 10)
swapstatus(base, rel, limit = 10)
swapstatus(requestid, quoteid, pending = 0, fast = 0)
recentswaps(limit = 3)
notarizations(coin)
public API: getcoins()
getcoin(coin)
portfolio()
getpeers()
passphrase(passphrase, gui, netid = 0, seednode = )
listunspent(coin, address)
setconfirms(coin, numconfirms, maxconfirms = 6)
trust(pubkey, trust)# positive to trust, 0 for normal, negative to␣
↪blacklist
balance(coin, address)
balances(address)
fundvalue(address = , holdings = [], divisor = 0)
orderbook(base, rel, duration = 3600)
getprices()
inuse()
getmyprice(base, rel)
getprice(base, rel)
```

(continues on next page)

(continued from previous page)

```
//sendmessage(base=coin, rel=, pubkey=zero, <argjson method2>)
//getmessages(firsti=0, num=100)
//deletemessages(firsti=0, num=100)
secretaddresses(prefix = 'secretaddress', passphrase, num = 10,␣
↪pubtype = 60, taddr = 0)
electrum(coin, ipaddr, port)
snapshot(coin, height)
snapshot_balance(coin, height, addresses[])
dividends(coin, height, < args > )
stop()
bot_list()
bot_statuslist()
bot_buy(base, rel, maxprice, relvolume) - > botid
bot_sell(base, rel, minprice, basevolume) - > botid
bot_settings(botid, newprice, newvolume)
bot_status(botid)
bot_stop(botid)
bot_pause(botid)
calcaddress(passphrase, coin = KMD)
convaddress(coin, address, destcoin)
instantdex_deposit(weeks, amount, broadcast = 1)
instantdex_claim()
timelock(coin, duration, destaddr = (tradeaddr), amount)
unlockedspend(coin, txid)
opreturndecrypt(coin, txid, passphrase)
getendpoint()
jpg(srcfile, destfile, power2 = 7, password, data = , required, ind␣
↪= 0)"}
```

5.2.13 jpg

barterDEX supports password encrypting data into a .jpg. The dest image is virtually indistinguishable from the original. Best to use a raw .jpg you created with a camera and not some file from the internet. As from the internet, it can be compared as to what bits are changed and in that case the encrypted rawdata will be visible.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"password\":\"123\",\
↪"ind\":3453,\"userpass\":\"$userpass\",\"method\":\"jpg\",\
↪"srcfile\":\"/root/boost_1_64_0/libs/gil/doc/doxygen/images/
↪monkey_steps.jpg\",\"destfile\":\"dest.jpg\",\"power2\":3,\"data\
↪":\"68656c6c6f20776f726c64\",\"required\":88}"
```

To write into a .jpg file, specify password, srcfile, destfile, data (hexstring) and required number of bits, power2. The same values of power2, password and required are needed to extract the data.

```
curl --url "http://127.0.0.1:7783" --data "{\"password\":\"123\",\
↪"userpass\":\"$userpass\",\"method\":\"jpg\",\"srcfile\":\"dest.
↪jpg\",\"power2\":3,\"required\":88}"
```

Just the name of the image, power2 are required along with the password.

Sample Output:

```
   ␣
↪47007d0d6e91a7844e14f685153c71f57b20f79b3d0204009eb1bcef0000000000000000000000
↪encoded .71
   ␣
↪47007d0d6e91a7844e14f685153c71f57b20f79b3d0204009eb1bcef0000000000000000000000
↪restored
    68656c6c6f20776f726c64 VERIFIED decryption .11 ind.3453 msglen .
↪71 required .568 New DCT coefficients successfully written to␣
↪dest.jpg, capacity 35672 modifiedrows .1 / 114 emit .568
    {
    "result": "success",
    "modifiedrows": 1,
    "outputfile": "dest.jpg",
    "power2": 3,
    "capacity": 35672,
    "required": 88,
    "ind": 65535,
    "decoded": "3a4e7dc2785e40d58b8784"
}
   ␣
↪47007d0d6e91a7844e14f685153c71f57b20f79b3d0204009eb1bcef0000000000000000000000
↪restored {
    "result": "success",
    "modifiedrows": 0,
    "power2": 3,
    "capacity": 35672,
    "required": 88,
    "ind": 3453,
    "decoded": "68656c6c6f20776f726c64"
}
```

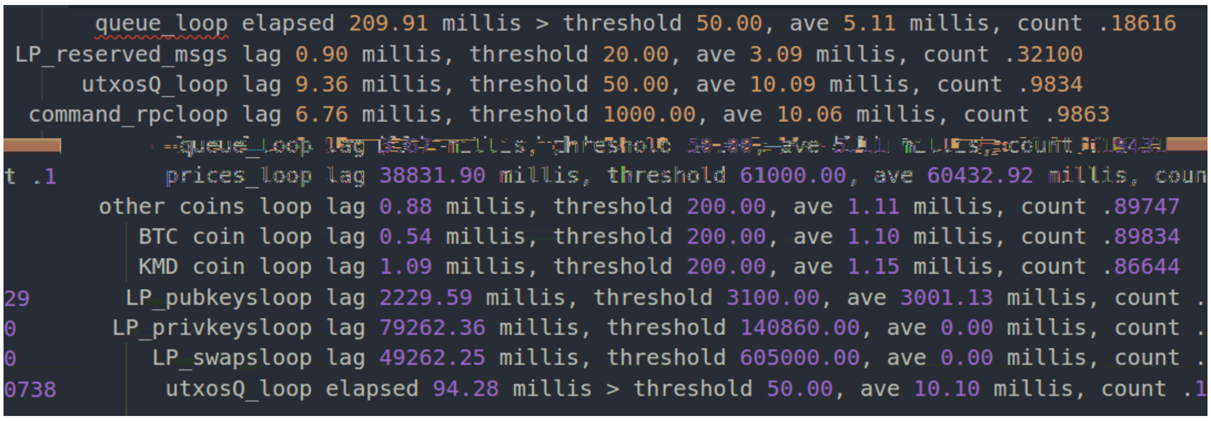
5.2.14 millis

`millis` is a new debug API. James put the millis display in the swaploop, which is once per 10 minutes.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"millis\"}"
```

Sample Output:



```
      queue_loop elapsed 209.91 millis > threshold 50.00, ave 5.11 millis, count .18616
 LP_reserved_msgs lag 0.90 millis, threshold 20.00, ave 3.09 millis, count .32100
     utxosQ_loop lag 9.36 millis, threshold 50.00, ave 10.09 millis, count .9834
  command_rpcloop lag 6.76 millis, threshold 1000.00, ave 10.06 millis, count .9863
               queue_loop lag [illegible] millis, threshold [illegible] ave [illegible] millis, count [illegible]
t .1             prices_loop lag 38831.90 millis, threshold 61000.00, ave 60432.92 millis, coun
         other coins loop lag 0.88 millis, threshold 200.00, ave 1.11 millis, count .89747
            BTC coin loop lag 0.54 millis, threshold 200.00, ave 1.10 millis, count .89834
            KMD coin loop lag 1.09 millis, threshold 200.00, ave 1.15 millis, count .86644
29          LP_pubkeysloop lag 2229.59 millis, threshold 3100.00, ave 3001.13 millis, count .
0          LP_privkeysloop lag 79262.36 millis, threshold 140860.00, ave 0.00 millis, count .
0             LP_swapsloop lag 49262.25 millis, threshold 605000.00, ave 0.00 millis, count .
0738           utxosQ_loop elapsed 94.28 millis > threshold 50.00, ave 10.10 millis, count .1
```

5.2.15 notarizations

This script will display coin notarization status for a specified coin. Use the script like this `./notarizations KMD`, `./notarizations REVS`, etc.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"notarizations\",\"coin\":\"$1\"}"
```

Sample Output:

```
{
    "result": "success",
    "coin": "KMD",
    "lastnotarization": 553573,
    "bestheight": 553578
}
```

5.2.16 parselog

`./parselog` will parse the stats.log file, just the incremental since last time.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"parselog\"}"
```

Sample Output:

```
{
    "result": "success",
    "newlines": 0,
    "request": 0,
    "reserved": 0,
    "connect": 0,
```

(continues on next page)

(continued from previous page)

```
    "connected": 0,
    "duplicates": 0,
    "parse_errors": 0,
    "uniques": 0,
    "tradestatus": 0,
    "unknown": 0
}
```

5.2.17 run

`./run` starts barterDEX in LP (Liquid Provider) mode while `./client` is for client mode. After it starts (takes a tiny bit time to complete) execute your API calls from the `dexscripts` directory. Don't use this API unless you have reliable internet connection from datacenter or vps.

Sample File Contents:

```
#!/bin/bash
source passphrase
source coins
./stop
git pull;
cd ..;
./m_mm;
pkill -15 marketmaker;
stdbuf -oL $1 ./marketmaker "{\"gui\":\"nogui\", \"profitmargin\":0.
↪01, \"userhome\":\"/${HOME#"/"}\", \"passphrase\":\"$passphrase\",
↪ \"coins\":$coins}" &
```

5.2.18 setpassphrase

This method helps the GUI build to take input of the passphrase and generate userpass. This is the second API to run in BarterDEX. On the first call it will display the userpass value at the top of output.

Sample File Content:

```
#!/bin/bash
source userpass
source passphrase
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\
↪"1d8b27b21efabcd96571cd56f91a40fb9aa4cc623d273c63bf9223dc6f8cd81f\
↪",\"method\":\"passphrase\",\"passphrase\":\"$passphrase\",\"gui\
↪":\"nogui\"}"
```

5.2.19 sleep

sleep API is to create a call with definite duration. What the below example scritps does is, it starts sleep async and it gets queued. While that is happening, we do 2 orderbooks, which immediately return. After 10 seconds the sleep is done and the normal command queue completes and the getcoin is executed. This changes the serialized nature of calls. So, it might destabilize things, but by being limited to orderbook and portfolio the risk is very small. New API can be added to the whitelisted fast calls, by testing it with a "fast":1.

Sample file content:

```
curl --url "http://127.0.0.1:7783" --data "{\"queueid\":1,\
↪"userpass\":\"$userpass\",\"method\":\"sleep\",\"seconds\":10}" &
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"orderbook\",\"base\":\"REVS\",\"rel\":\
↪"KMD\"}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"orderbook\",\"base\":\"REVS\",\"rel\":\
↪"KMD\"}"
curl --url "http://127.0.0.1:7783" --data "{\"queueid\":2,\
↪"userpass\":\"$userpass\",\"method\":\"getcoin\",\"coin\":\"REVS\
↪",\"rel\":\"KMD\"}"
```

5.2.20 stop

This method will stop barterDEX.

Sample File Content:

```
#!/bin/bash
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"stop\"}"
```

5.2.21 trust

Make your pubkey trusted by using this ./trust method. You need to add your pubkey after the command. If you want the pubkey to be not trusted use “trust”:-1. -1 means dont trust, 1 means to trust. This API works immediately.

Sample File Contents:

```
echo "usage: ./trust <pubkey>"
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"pubkey\":\"$1\",\"method\":\"trust\",\"trust\":1}"
```

Sample Output:

```
usage: ./trust <pubkey>
{"result":"success"}
```

5.2.22 trusted

This method will display the `pubkey` you made trusted. No need to edit the file.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"trusted\"}"
```

Sample Output:

```
[

↪"040f1d2d5d12027afa2cec30477312e225b0d24c77cc4aa08d3fffe51277b904"
]
```

5.3 barterDEX Trading

Default timeout for a trade is 10 seconds, which means if no response, must wait 10 seconds between trade requests. It will generate an error if Alice tries to submit a trade while a previous request is pending.

However if the other side responds, you can do another trade and we are seeing virtually instant responses from the live LP nodes.

5.3.1 Trade Negotiation Sequence:

```
Alice submits a "request" to the Bob node.
Now we are in a pending state for up to 10 seconds.
Bob responds with a "reserved", which releases Alice from the␣
↪pending state.
For the request that comes back, Alice can reject it or accept it␣
↪and send a "connect" message.
Finally Bob returns a "connected" message and the atomic swap␣
↪begins.
```

James has also added automated broadcast of any setprices, which will occur automatically when you do a buy/sell for the coin you are buying with, as long as you are not using Electrum.

To be a bob, you need the native coin. With the pruning of the orderbook to most recent 2 minutes, it required the setprice to be called regularly.

This function is internalized, so a single setprice is all that is needed. If you want to "cancel" it you can setprice to 0. GUI can now post bob orders if you have native coins enabled.

Using the `buy/sell` api is a fill or kill (except partial fills are allowed) and to put a limit order, `autoprice` needs to be used. The `autoprice` is a bit tricky to use, make sure you don't make the example backwards.

Note: To fully cancel an autotrade, `setprice 0` needs to be called twice, once with `base/rel` and then with `rel/base`, since there are actually 2 prices (bid and ask).

5.3.2 autoprice

`autoprice` is a very powerful API and it allows you to specify the price for a specific trading pair that is automatically updated as the market price of it changes. barterDEX uses external price sources (Bittrex, Cryptopia, Coinmarketcap) to get up to date prices. You can use fixed price instead of `margin` or use autoprice based on coinmarketcap. For now, it is a relatively simple set of things you can do with the following fields (`base, rel, fixed, minprice, maxprice, margin, refbase, refrel, factor, offset`).

Sample File Contents:

```
#!/bin/bash
margin=0.05
source userpass

# KMD/BTC must be first as other prices depend on it
#curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"autoprice\",\"base\":\"KMD\",\"rel\":\
↪"BTC\",\"margin\":$margin,\"refbase\":\"komodo\",\"refrel\":\
↪"coinmarketcap\"}"
#curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"autoprice\",\"base\":\"BTC\",\"rel\":\
↪"KMD\",\"fixed\":0.00025,\"margin\":$margin}"
#curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"autoprice\",\"base\":\"KMD\",\"rel\":\
↪"BTC\",\"fixed\":4000,\"margin\":$margin}"
curl --url "http://127.0.0.1:7783" --data "{\"minprice\":0.0003,\
↪"maxprice\":0.001,\"userpass\":\"$userpass\",\"method\":\
↪"autoprice\",\"base\":\"KMD\",\"rel\":\"BTC\",\"margin\":0.05,\
↪"refbase\":\"komodo\",\"refrel\":\"coinmarketcap\"}"

./auto_chipskmd
./auto_chipsbtc

#curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"autoprice\",\"base\":\"KMD\",\"rel\":\
↪"MNZ\",\"offset\":0.0,\"refbase\":\"KMD\",\"refrel\":\"BTC\",\
↪"factor\":15000,\"margin\":-0.2}"

curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"autoprice\",\"base\":\"HUSH\",\"rel\":\
↪"KMD\",\"margin\":$margin,\"refbase\":\"hush\",\"refrel\":\
↪"coinmarketcap\"}"
```

(continues on next page)

(continued from previous page)

```
#curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"autoprice\",\"base\":\"KMD\",\"rel\":\
↪"BTCH\",\"offset\":0.0,\"refbase\":\"KMD\",\"refrel\":\"HUSH\",\
↪"factor\":1.44,\"buymargin\":0.05,\"sellmargin\":0.05}"
#curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"autoprice\",\"base\":\"BTCH\",\"rel\":\
↪"KMD\",\"offset\":0.0,\"refbase\":\"HUSH\",\"refrel\":\"KMD\",\
↪"factor\":0.7,\"buymargin\":0.05,\"sellmargin\":0.05}"

curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"autoprice\",\"base\":\"BEER\",\"rel\":\
↪"PIZZA\",\"fixed\":0.0001,\"margin\":0.00001}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"autoprice\",\"base\":\"BEER\",\"rel\":\
↪"ETOMIC\",\"fixed\":10,\"margin\":0.00001}"

source crypto
source trackbtc

#source jumblr
#source trackbtc

source pangea
source trackbtc

source bet
source trackbtc

#source revs
#source trackbtc

sharkholdings="{\"coin\":\"iota\",\"balance\":1500000}, {\"coin\":\
↪"komodo\",\"balance\":120000}, {\"coin\":\"bitcoin-cash\",\
↪"balance\":1200}, {\"coin\":\"bitcoin\",\"balance\":100}"
curl --url "http://127.0.0.1:7783" --data "{\"base\":\"MSHARK\",\
↪"rel\":\"KMD\",\"fundvalue_bid\":\"NAV_KMD\",\"fundvalue_ask\":\
↪"assetNAV_KMD\",\"userpass\":\"$userpass\",\"method\":\"autoprice\
↪",\"margin\":$margin,\"address\":\
↪"RTu3JZZKLJTcfNwBa19dWRagEfQq49STqC\",\"holdings\":[
↪$sharkholdings],\"divisor\":1400000}"

curl --url "http://127.0.0.1:7783" --data "{\"margin\":$margin,\
↪"base\":\"SUPERNET\",\"rel\":\"KMD\",\"fundvalue_bid\":\"NAV_KMD\
↪",\"fundvalue_ask\":\"NAV_KMD\",\"userpass\":\"$userpass\",\
↪"method\":\"autoprice\",\"address\":\
↪"RRyyejME7LRTuvdziWsXkAbSW1fdiohGwK\",\"holdings\":[{\"coin\":\
↪"iota\",\"balance\":11000000}, {\"coin\":\"stratis\",\"balance\
↪":1300000}, {\"coin\":\"zcash\",\"balance\":0.10000}, {\"coin\":\
↪"syscoin\",\"balance\":20000000}, {\"coin\":\"waves\",\"balance\
↪":700000}, {\"coin\":\"bitcoin\",\"balance\":600}, {\"coin\":\
↪"bitcoin-cash\",\"balance\":1500}, {\"coin\":\"heat-ledger\",\
↪"balance\":2323851 }, {\"coin\":\"decred\",\"balance\":0.20000},
↪{\"coin\":\"vericoin\",\"balance\":2199368 }, {\"coin\":\
↪"byteball\",\"balance\":4238}, {\"coin\":\"iocoin\",\"balance\":0.
↪150000}, {\"coin\":\"quantum-resistant-ledger\",\"balance\":0.
```

(continues on next page)

(continued from previous page)

```
curl --url "http://127.0.0.1:7783" --data "{\"margin\":$margin,\
↪"base\":\"HODL\",\"rel\":\"KMD\",\"fundvalue_bid\":\"assetNAV_KMD\
↪",\"fundvalue_ask\":\"assetNAV_KMD\",\"userpass\":\"$userpass\",\
↪"method\":\"autoprice\",\"address\":\
↪"RNcUaMUEFLxVwtTo7rgruhwYanGk1jTkeU\",\"holdings\":[{\"coin\":\
↪"siacoin\",\"balance\":185000000,\"comment\":\"using siafunds␣
↪equal to million siacoin\"}],\"divisor\":10000000}"

dexholdings="{\"coin\":\"blocknet\",\"balance\":2500000}"

curl --url "http://127.0.0.1:7783" --data "{\"base\":\"DEX\",\"rel\
↪":\"KMD\",\"margin\":$margin,\"fundvalue_bid\":\"assetNAV_KMD\",\
↪"fundvalue_ask\":\"assetNAV_KMD\",\"userpass\":\"$userpass\",\
↪"method\":\"autoprice\",\"address\":\
↪"RThtXup6Zo7LZAi8kRWgjAyi1s4u6U9Cpf\",\"holdings\":[$dexholdings],
↪\"divisor\":1000000}"

curl --url "http://127.0.0.1:7783" --data "{\"base\":\"BOTS\",\"rel\
↪":\"KMD\",\"margin\":$margin,\"fundvalue_bid\":\"assetNAV_KMD\",\
↪"fundvalue_ask\":\"assetNAV_KMD\",\"userpass\":\"$userpass\",\
↪"method\":\"autoprice\",\"address\":\
↪"RNdqHx26GWy9bk8MtmH1UiXjQcXE4RKK2P\",\"holdings\":[$dexholdings],
↪\"divisor\":3333333}"

curl --url "http://127.0.0.1:7783" --data "{\"base\":\"JUMBLR\",\
↪"rel\":\"KMD\",\"margin\":$margin,\"fundvalue_bid\":\"assetNAV_
↪KMD\",\"fundvalue_ask\":\"assetNAV_KMD\",\"userpass\":\"$userpass\
↪",\"method\":\"autoprice\",\"address\":\
↪"RGhxXpXSSBTBm9EvNsXnTQczthMCxHX91t\",\"holdings\":[$dexholdings],
↪\"divisor\":3333333}"

curl --url "http://127.0.0.1:7783" --data "{\"base\":\"MGW\",\"rel\
↪":\"KMD\",\"margin\":$margin,\"fundvalue_bid\":\"assetNAV_KMD\",\
↪"fundvalue_ask\":\"assetNAV_KMD\",\"userpass\":\"$userpass\",\
↪"method\":\"autoprice\",\"holdings\":[$dexholdings],\"divisor\
↪":13000000}"

curl --url "http://127.0.0.1:7783" --data "{\"base\":\"REVS\",\"rel\
↪":\"KMD\",\"margin\":$margin,\"fundvalue_bid\":\"assetNAV_KMD\",\
↪"fundvalue_ask\":\"assetNAV_KMD\",\"userpass\":\"$userpass\",\
↪"method\":\"autoprice\",\"holdings\":[$dexholdings],\"divisor\
↪":9000000}"
```

Sample Output:

```
{
            "result": "success"
},
```

(continues on next page)

(continued from previous page)

```
{
        "result": "success"
}
.......
```

Knowledge Base:

`refbase` and `refrel` allow you to specify a different base/rel pair as the basis for the price, if there is no `refbase` and `refrel`, the the base/rel price from external sources is used as the starting point. This is calculated approximately once per minute and a set price done automatically:

```
setprice of (1. + margin) * ((price * factor) + offset)
```

The margin is usually set to 0.01 for 1% profit margin. using factor and offset it is possible to map a starting price to a standard multiple.

There is also a minprice field which sets the absolute minimum (post calculation) price that is accepted.

Example `autoprice` script using coinmarketcap prices.

```
curl --url "http://127.0.0.1:7783" --data "{\"minprice\":0.00002,\
↪"maxprice\":0.0001,\"userpass\":\"$userpass\",\"method\":\
↪"autoprice\",\"base\":\"CHIPS\",\"rel\":\"BTC\",\"margin\":0.05,\
↪"refbase\":\"chips\",\"refrel\":\"coinmarketcap\"}"
```

or

```
curl --url "http://127.0.0.1:7783" --data "{\"minprice\":0.04,\
↪"maxprice\":0.1,\"userpass\":\"$userpass\",\"method\":\"autoprice\
↪",\"base\":\"CHIPS\",\"rel\":\"KMD\",\"margin\":0.05,\"refbase\":\
↪"chips\",\"refrel\":\"coinmarketcap\"}"
```

`autoprice` the value of a fund, ie. using the `fundvalue` api

The way to do is it start with the `fundvalue` api call, then change the `method` to `autoprice`. Add `base` and `rel` and most importantly `fundvalue_bid` and `fundvalue_ask` that will be values in the return of the fundvalue call. Additionally, `margin` is applied. This sounds complicated and it is, but now MSHARK, HODL and even SUPERNET are running using this autoprice. The following sample content is an example how to use it.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"base\":\"MSHARK\",\
↪"rel\":\"KMD\",\"fundvalue_bid\":\"NAV_KMD\",\"fundvalue_ask\":\
↪"assetNAV_KMD\",\"userpass\":\"$userpass\",\"method\":\"autoprice\
↪",\"address\":\"RTu3JZZKLJTcfNwBa19dWRagEfQq49STqC\",\"holdings\
↪":[{\"coin\":\"iota\",\"balance\":5000000}],\"divisor\":1400000}"
```

Sample Output:

```
{
        "result":"success"
}
```

autoprice using usdpeg

There is a way for autoprice coinmarketcap to do USD pegs for orderbooks denominated in any coin. The syntax is pretty arbitrary but it works.

Make sure the `usdpeg` is set to non-zero. In that case put in the orderbook BACKWARDS (dont ask me why, it just works only in this way) for the coin that is the coinmarketcap refbase. Based on the example it is KMD so base KMD and rel OOT, will create OOT/KMD orderbooks using factor 0.15 (fifteen cents) denominated in KMD. The example script used different margins for buymargin and sellmargin, so it works for dICO scenario (allow people to sell back but at a steep loss) and also it is backwards (dont ask me why).

What this means is if you do a base DOGE rel OOT and refbase dogecoin, it should maintain an orderbook in DOGE pegged to the 15 cents price. Any CMC coin can be used. If you want to change the price, change the value of `factor` accordingly.

Sample File Content:

```
#!/bin/bash
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"autoprice\",\"base\":\"KMD\",\"rel\":\
↪"OOT\",\"factor\":0.15,\"buymargin\":0.0001,\"sellmargin\":0.2,\
↪"refbase\":\"komodo\",\"refrel\":\"coinmarketcap\",\"usdpeg\":1}"
```

Sample Output:

```
{
        "result":"success"
}
```

5.3.3 buy

In simple terms, this command will post a buy order. If there is a matching order in the orderbooks, then this will automatically start a trade. If a match is not found, a bid order will be added in the orderbook. Buy fills asks from the orderbook using relvolume. Trade request will last as long as defined in the 'timeout' field.

Call 'recentswaps' to check the status of the trade request. Alice side does not give feedback about expired trade request, unless a 'recentswaps' call is done.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"buy\",\"base\":\"MNZ\",\"rel\":\"KMD\",\
↪"relvolume\":0.005,\"price\":0.172}"
```

Knowledge Base:

`base`: the currency you want to buy `rel`: the currency you are paying with `price`: the max price you are willing to pay for 1 base `relvolume`: the amount of rel you want to trade `timeout`: the amount of time Alice or Bob should wait for payment. Default: 10 seconds (might be deprecated in future) `duration`: after the specified duration it wont be displayed

Sample Output:

```
{
    "result": "success",
    "swaps": [
        [
            45030989,
            41860326
        ],
        [
            264022270,
            1424922977
        ],
        [
            3882191272,
            3366340834
        ]
    ],
    "pending": {
        "expiration": 1509564162,
        "timeleft": 15,
        "requestid": 0,
        "quoteid": 0,
        "bob": "MNZ",
        "base": "MNZ",
        "basevalue": 0.02803687,
        "alice": "KMD",
        "rel": "KMD",
        "relvalue": 0.0048,
        "aliceid": 7785695213943587000
```

(continues on next page)

(continued from previous page)

```
    }
}
```

5.3.4 electrum

This method enables a coin by connecting to the specified electrumx server. For this method running a native node and blockchain download is not necessary. You need to edit the file with coin name and IP address and port for the coin. For a list of electrumx server check this link http://pad.supernet.org/electrum-servers

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"electrum\",\"coin\":\"KMD\",\"ipaddr\":\
↪"electrum1.cipig.net\",\"port\":10001}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"electrum\",\"coin\":\"MNZ\",\"ipaddr\":\
↪"electrum1.cipig.net\",\"port\":10002}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"electrum\",\"coin\":\"REVS\",\"ipaddr\":\
↪"electrum1.cipig.net\",\"port\":10003}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"electrum\",\"coin\":\"JUMBLR\",\"ipaddr\
↪":\"electrum1.cipig.net\",\"port\":10004}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"electrum\",\"coin\":\"SUPERNET\",\
↪"ipaddr\":\"electrum1.cipig.net\",\"port\":10005}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"electrum\",\"coin\":\"DEX\",\"ipaddr\":\
↪"electrum1.cipig.net\",\"port\":10006}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"electrum\",\"coin\":\"BOTS\",\"ipaddr\":\
↪"electrum1.cipig.net\",\"port\":10007}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"electrum\",\"coin\":\"CRYPTO\",\"ipaddr\
↪":\"electrum1.cipig.net\",\"port\":10008}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"electrum\",\"coin\":\"HODL\",\"ipaddr\":\
↪"electrum1.cipig.net\",\"port\":10009}"
```

5.3.5 getfee

`getfee` API will display network tx fee based on that time. This should only be required for BTC, all other coins are hardcoded.

Sample file content:

```
#!/bin/bash
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":       \"
↪$userpass\",\"method\":\"getfee\",\"coin\":\"BTC\"}"
```

Sample file output:

```
{
        "result":"success",
        "coin":"BTC",
        "txfee":0.00020000
}
```

5.3.6 getprices

The `./getprices` API lists all currently available trading pairs and its current prices. You need to specify a `base` and `rel` coin if you need a specific pair price.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"getprices\",\"coin\":\"KMD\"}"
```

Sample Output:

```
[{
    "pubkey":
↪"040f1d2d5d12027afa2cec30477312e225b0d24c77cc4aa08d3fffe51277b904
↪",
    "rmd160": "8784673d815a20300412a9010b72e771963d45b6",
    "pubsecp":
↪"03829863f4ed7660fbf62b2f84b0e655f523a33f23a1f681107d2a2942987584b1
↪",
    "timestamp": 1508160175,
    "asks": [
        [
            "BTC",
            "KMD",
            3304.53174512
        ],
        [
            "KMD",
            "BTC",
            0.00030268
        ],
        [
            "KMD",
            "REVS",
            0.78551134
```

(continues on next page)

(continued from previous page)

```
        ],
        [
            "KMD",
            "JUMBLR",
            0.10432527
        ],
        [
            "KMD",
            "HUSH",
            1.17931676
        ],
        [
            "REVS",
            "KMD",
            1.29051134
        ],
        [
            "JUMBLR",
            "KMD",
            9.77904977
        ],
        [
            "HUSH",
            "KMD",
            0.86507888
        ]
    ]
}]
```

5.3.7 goal

The `./goal` command is basically the automatic trading tool of the API. When you execute this command, orders will be set until the goals are reached. The idea is that you set the goal for each coin and if you have more than the goal for a specific coin, it will allow sales of that coin. If you have less than the goal percentage, it will allow buys of that coin.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"goal\",\"coin\":\"KMD\",\"val\":99}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"goal\",\"coin\":\"BTC\",\"val\":10}"
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"goal\",\"coin\":\"REVS\",\"val\":1}"
```

Sample Output:

```
{
    "result": "success",
    "kmd_equiv": 29.03290329,
    "buycoin": "KMD",
    "buyforce": 125.75072424,
    "sellcoin": "DGB",
    "sellforce": -412.28377122,
    "portfolio": [{
            "coin": "BTC",
            "address": "15XAhHK4ULofD6BTckrCWK6XSkCjscX7jj",
            "amount": 0,
            "price": 3315.35638637,
            "kmd_equiv": 0,
            "perc": 0,
            "goal": 10,
            "goalperc": 9.09090909,
            "relvolume": 0,
            "force": 82.6446281,
            "balanceA": 0,
            "valuesumA": 0,
            "balanceB": 0,
            "valuesumB": 0,
            "balance": 0
        },
        {
            "coin": "KMD",
            "address": "RDoMmoCM5AcEH6Yf5vqKbqRjD1fLcQHuWb",
            "amount": 22.87390295,
            "price": 1,
            "kmd_equiv": 22.87390295,
            "perc": 78.78613696,
            "goal": 99,
            "goalperc": 90,
            "relvolume": 0,
            "force": 125.75072424,
            "balanceA": 22.6448,
            "valuesumA": 22.87390295,
            "aliceutil": 98.99840901,
            "balanceB": 22.6448,
            "valuesumB": 22.87390295,
            "balance": 22.87390295,
            "bobutil": 98.99840901
        },
        {
            "coin": "DGB",
            "address": "D9fGEYFhmkhwk6N4MLqm45G8Ksw3E2AmTR",
            "amount": 919,
            "price": 0.00670185,
            "kmd_equiv": 6.15900034,
            "perc": 21.21386304,
```

(continues on next page)

(continued from previous page)

```
                "goal": 1,
                "goalperc": 0.90909091,
                "relvolume": 186.60085587,
                "force": -412.28377122,
                "balanceA": 479.499,
                "valuesumA": 919,
                "aliceutil": 52.17616975,
                "balanceB": 479.499,
                "valuesumB": 919,
                "balance": 919,
                "bobutil": 52.17616975
        }
    ]
}
```

5.3.8 goals

Displays the goal percentage for each coin and their status.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"goal\"}"
```

Sample Output:

```
{
    "result": "success",
    "kmd_equiv": 28.98048555,
    "buycoin": "BTC",
    "buyforce": 625,
    "sellcoin": "KMD",
    "sellforce": -2908.30091422,
    "base": "BTC",
    "rel": "KMD",
    "relvolume": 12.33191136,
    "portfolio": [{
            "coin": "BTC",
            "address": "15XAhHK4ULofD6BTckrCWK6XSkCjscX7jj",
            "amount": 0,
            "price": 3321.5219084,
            "kmd_equiv": 0,
            "perc": 0,
            "goal": 25,
            "goalperc": 25,
            "relvolume": 0,
            "force": 625,
            "balanceA": 0,
```

(continues on next page)

(continued from previous page)

```
            "valuesumA": 0,
            "balanceB": 0,
            "valuesumB": 0,
            "balance": 0
        },
        {
            "coin": "KMD",
            "address": "RDoMmoCM5AcEH6Yf5vqKbqRjD1fLcQHuWb",
            "amount": 22.87391037,
            "price": 1,
            "kmd_equiv": 22.87391037,
            "perc": 78.92866505,
            "goal": 25,
            "goalperc": 25,
            "relvolume": 12.33559451,
            "force": -2908.30091422,
            "balanceA": 22.6448,
            "valuesumA": 22.87391037,
            "aliceutil": 98.9983769,
            "balanceB": 22.6448,
            "valuesumB": 22.87391037,
            "balance": 22.87391037,
            "bobutil": 98.9983769
        },
        {
            "coin": "ZEC",
            "address": "t1NPmhcjCSfbFojEMZBfKe8CShQPpkcn4W7",
            "amount": 0,
            "price": 135.89703925,
            "kmd_equiv": 0,
            "perc": 0,
            "goal": 25,
            "goalperc": 25,
            "relvolume": 0,
            "force": 625,
            "balanceA": 0,
            "valuesumA": 0,
            "balanceB": 0,
            "valuesumB": 0,
            "balance": 0
        },
        {
            "coin": "DGB",
            "address": "D9fGEYFhmkhwk6N4MLqm45G8Ksw3E2AmTR",
            "amount": 919,
            "price": 0.0066448,
            "kmd_equiv": 6.10657518,
            "perc": 21.07133495,
            "goal": 25,
```

(continues on next page)

(continued from previous page)

```
                "goalperc": 25,
                "relvolume": 0,
                "force": 15.43440909,
                "balanceA": 479.499,
                "valuesumA": 919,
                "aliceutil": 52.17616975,
                "balanceB": 479.499,
                "valuesumB": 919,
                "balance": 919,
                "bobutil": 52.17616975
            }
    ]
}
```

5.3.9 myprice

Shows number of bids andprices you set for a given coin. Your wallet must be active to display the prices.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"myprice\",\"base\":\"REVS\",\"rel\":\
↪"KMD\"}"
```

Sample Output:

```
{
  "base": "REVS",
  "rel": "KMD",
  "bid": 2,
  "ask": 0
}
```

5.3.10 myprices

`./myprices` shows numer of bids and ask for your `./buy` coin pair.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"myprices\"}"
```

Sample Output:

```
[{
        "base": "KMD",
```

(continues on next page)

(continued from previous page)

```
        "rel": "REVS",
        "bid": 0,
        "ask": 0.5
    },
    {
        "base": "REVS",
        "rel": "KMD",
        "bid": 2,
        "ask": 0
    }
]
```

5.3.11 orderbook

Displays bid/ask for a specific coin/asset pair. You can display multiple pairs, just need to edit the file and add more line. The `numutxos/minvolume/maxvolume` are the state of your local `utxo` cache. It might or might not be 100% accurate. Also, if all `0`'s it could be that it hasn't been queried yet. Even if all zeros, it is possible, even likely that there are actually `utxo` available.

If you do an `./orderbook`, it will start fetching the details from the best prices. You can do `listunspent` for a specific address, but it is not possible to know for sure what trades are possible, especially as there are other nodes that might be going after the same `utxo`. Specifically doing a `listunspent` api will also fetch the details, also doing a trade will fetch the details as well. James uses just in time fetching to get the `utxo` data. This way we don't flood the network with data for `utxo` info.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"orderbook\",\"base\":\"DGB\",\"rel\":\
↪"KMD\"}"
```

Sample Output:

```
{
    "bids": [
        {
            "coin": "KMD",
            "address": "RB8yufv3YTfdzYnwz5paNnnDynGJG6WsqD",
            "price": 0.00492739,
            "numutxos": 13,
            "minvolume": 0.072,
            "maxvolume": 0.76,
            "pubkey":
↪"8adbf9d5de4fb49ec1fca9ca3f28ba384715752de1197c9cddbb756c3d2a1a7c
↪",
            "age": 16
```

(continues on next page)

(continued from previous page)

```
        },
        {
            "coin": "KMD",
            "address": "RHPBQQpjVqpwWC8NrpAxdoRQAkigA2zjRo",
            "price": 0.00487791,
            "numutxos": 5,
            "minvolume": 0.00016002,
            "maxvolume": 96,
            "pubkey":
↪"22aaf8e3686d31ee9f5ebe61c97231114717aee943ef01cc58e9f843ddad0240
↪",
            "age": 53
        }
    ],
    "numbids": 2,
    "asks": [
        {
            "coin": "DGB",
            "address": "D6ztNQyQF3mMTYcMFVq1q2cd6eXzqPKGc8",
            "price": 0.00532946,
            "numutxos": 38,
            "minvolume": 0.00586622,
            "maxvolume": 2.20800448,
            "pubkey":
↪"8adbf9d5de4fb49ec1fca9ca3f28ba384715752de1197c9cddbb756c3d2a1a7c
↪",
            "age": 16
        },
        {
            "coin": "DGB",
            "address": "DDF5s9t6CRveyBwn8EBQ63FoHczNsyFsFi",
            "price": 0.0053779,
            "numutxos": 3,
            "minvolume": 0.67844886,
            "maxvolume": 7.06141621,
            "pubkey":
↪"22aaf8e3686d31ee9f5ebe61c97231114717aee943ef01cc58e9f843ddad0240
↪",
            "age": 53
        }
    ],
    "numasks": 2,
    "base": "DGB",
    "rel": "KMD",
    "timestamp": 1508410319
}
```

5.3.12 portfolio

Shows all active coin information and value. Such as, smartaddress, balance, current price per coin in barterDEX, KMD equivalent balance, percentage of your portfolio goal. You don't need to edit or change the script for it work. It will get information from your active wallets and display them.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"portfolio\"}"
```

Sample Output:

```
{
    "result": "success",
    "kmd_equiv": 29.09215302,
    "buycoin": "KMD",
    "buyforce": 129.37341083,
    "sellcoin": "DGB",
    "sellforce": -418.82246271,
    "base": "ZEC",
    "rel": "DGB",
    "relvolume": 189.62190318,
    "portfolio": [
        {
            "coin": "BTC",
            "address": "15XAhHK4ULofD6BTckrCWK6XSkCjscX7jj",
            "amount": 0,
            "price": 3351.92820521,
            "kmd_equiv": 0,
            "perc": 0,
            "goal": 10,
            "goalperc": 9.09090909,
            "relvolume": 0,
            "force": 82.6446281,
            "balanceA": 0,
            "valuesumA": 0,
            "balanceB": 0,
            "valuesumB": 0,
            "balance": 0
        },
        {
            "coin": "KMD",
            "address": "RDoMmoCM5AcEH6Yf5vqKbqRjD1fLcQHuWb",
            "amount": 22.87392545,
            "price": 1,
            "kmd_equiv": 22.87392545,
            "perc": 78.62575669,
            "goal": 99,
            "goalperc": 90,
```

(continues on next page)

(continued from previous page)

```
                "relvolume": 0,
                "force": 129.37341083,
                "balanceA": 22.6448,
                "valuesumA": 22.87392545,
                "aliceutil": 98.99831163,
                "balanceB": 22.6448,
                "valuesumB": 22.87392545,
                "balance": 22.87392545,
                "bobutil": 98.99831163
            },
            {
                "coin": "DGB",
                "address": "D9fGEYFhmkhwk6N4MLqm45G8Ksw3E2AmTR",
                "amount": 919,
                "price": 0.0067663,
                "kmd_equiv": 6.21822757,
                "perc": 21.37424331,
                "goal": 1,
                "goalperc": 0.90909091,
                "relvolume": 188.07475055,
                "force": -418.82246271,
                "balanceA": 479.499,
                "valuesumA": 919,
                "aliceutil": 52.17616975,
                "balanceB": 479.499,
                "valuesumB": 919,
                "balance": 919,
                "bobutil": 52.17616975
            }
        ]
}
```

5.3.13 sell

This script is completely opposite of `buy` and uses the same parameters. Edit the file with specific coin pair name, volume and price.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"sell\",\"base\":\"KMD\",\"rel\":\"BTC\",\
↪"basevolume\":10.0,\"price\":0.0005}"
```

Sample Output:

```
{
        "result": "success"
}
```

5.3.14 setconfirms

This will allow each node to set the number of required confirms on a coin by coin basis. During the atomic swap, each side sends to the other the required confirms and the "protocol" is that both sides use the bigger value.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"setconfirms\",\"coin\":\"DGB\",\
↪"numconfirms\":1}"
```

Sample Output:

```
{
            "result": "success"
}
```

5.3.15 setprice

To create a bob `utxo` (or an ask order) you need to first set the price. To set price you need to edit the `./setprice` script in the `dexscripts` folder

Note: To fully cancel an autotrade, `setprice 0` needs to be called twice, once with `base/rel` and then with `rel/base`, since there are actually 2 prices (bid and ask). Doing `setprice 0` for an order made with `coinmarketcap api` call doesn't end it. It is likely as it has two `0`'s and you can only set one `0` at a time. The only way to stop an order made with `coinmarketcap api` is to kill `marketmaker`.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"setprice\",\"base\":\"KMD\",\"rel\":\
↪"REVS\",\"price\":0.5}"
```

Sample Output:

```
{
          "result": "success"
}
```

5.3.16 fomo

`fomo` Fear of missing out API will allow you buy for your specified relvol at the best current orderbook price. Orderbook has to be visible for this API to work. When using `fomo` you won't get the best price. No need to `fomo` if you are able to do normal ordermatch. It adds 5% to the orderbook entry that is the biggest that fits.

Sample file content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"buy\",\"base\":\"KMD\",\"rel\":\"HODL\",\
↪"fomo\":40.00038}"
```

5.3.17 dump

As the name suggests, you can use `dump` API to dump the coins you want to at current order-book price. Orderbook has to be visible for this API to work.

Sample file content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"sell\",\"base\":\"KMD\",\"rel\":\"HODL\",
↪\"dump\":40.00038}"
```

5.3.18 pubkeystats

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"statsdisp\",\"starttime\":0,\"endtime\
↪":0,\"pubkey\":\
↪"a2593155464e37fcc88245780240a412a38cf3d316809445aad73f4e7789187d\
↪"}"
```

5.4 Status / Info

5.4.1 getendpoint

There is a new websockets realtime events to the barterDEX API family, `getendpoint`. It returns a nanomsg NN_PAIR endpoint that if you connect to it. You get an event stream, with hopefully all the relevant events, especially about the change in state of swap. The even stream is started as soon as the `getendpoint` API is called.

There is also a new mode for marketmaker: `./marketmaker events` which we should run in a new terminal. This should connect to the endpoint created by `getendpoint` API call. This will simply launch a new process and printout all the events. It will be like the console spew of printouts so you might want to `nohup` it and then `grep` to find things. This allows GUI to be done in an event driven way. Any command can add a `"queueid":nn` with a non-zero `nn` and it will queue the command. Its completion will arrive in the nanomsg event stream with the `queueid` and the result.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"getendpoint\"}"
```

Sample Output:

```
{
    "result": "success",
    "endpoint": "ws://127.0.0.1:5555",
    "socket": 20,
    "sockopt": 0
}
```

5.4.2 pendings

`./pendings` is a special case of `statsdisp` and will display all the pending swaps. Using `recentswaps` API is better suitable for most usecases.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"statsdisp\",\"starttime\":2000000000,\
↪"endtime\":2000000000}"
```

Sample Output:

```
{
    "result": "success",
    "newlines": 0,
    "swaps": [
        {
            "aliceid": "7276253190735921153",
            "src":
↪"30eb3d952f66eba2df0100b97d6707257cf91dc8c8b0237cfb7c3aa70dc52d4d
↪",
            "base": "REVS",
            "basevol": 0.13338748,
            "dest":
↪"1282417c4e7d5bb870067f5c05ffe87a0d8c8ce9956befff560e92b7c8d23741
↪",
            "rel": "KMD",
            "relvol": 0.1505,
            "price": 1.12913831,
            "requestid": 2339417255,
            "quoteid": 1177391651,
            "line": "2017-10-30T16:16:00Z (nogui).(nogui) 0   ␣
↪connected    7276253190735921153: (0.13338748  REVS) -> (0.
↪15050000   KMD) 1.12913831 finished.0 expired.0"
        }
    ],
```

(continues on next page)

(continued from previous page)

```
    "volumes": [
        {
            "coin": "KMD",
            "srcvol": 0,
            "destvol": 0.1505,
            "numtrades": 1,
            "total": 0.1505
        },
        {
            "coin": "REVS",
            "srcvol": 0.13338748,
            "destvol": 0,
            "numtrades": 1,
            "total": 0.13338748
        }
    ],
    "request": 4,
    "reserved": 2,
    "connect": 4,
    "connected": 1,
    "duplicates": 4,
    "parse_errors": 0,
    "uniques": 1,
    "tradestatus": 0,
    "unknown": 0
}
```

5.4.3 swapstatus

This will display the swap status for your trades along with `requestid` and `quoteid`. This API also helps unstuck swaps and help claim funds if the swap wasn't successfully completed. In order to claim stuck funds, remove all `.finished` file, start barterDEX, add coins and issue this script.

These available API options are described below:

```
swapstatus(pending=0)
swapstatus(coin, limit=10)
swapstatus(base, rel, limit=10)
swapstatus(requestid, quoteid, pending=0)
```

The base/rel swapstatus will scan ALL your historical swaps and that could take a very long time. There is a new parameter for the `swapstatus` API, `fast:1`. You can use this flag with swapstatus for `base/rel`, `requestid/quoteid` for faster and optimised performance while getting the response. 2 different example scripts:

```
curl --url "http://127.0.0.1:7783" --data "{\"pending\":1,\
↪"userpass\":\"$userpass\",\"method\":\"swapstatus\", \"base\":\
↪"CHIPS\", \"rel\":\"KMD\", \"fast\":1}"
```

(continues on next page)

(continued from previous page)

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"swapstatus\",\"requestid\":2291973695,\
↪"quoteid\":3387529385}, \"fast\":1}"
```

This is very helpful when you have so many swap files in DB/SWAPS dir.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"swapstatus\"}"
```

Sample Output:

```
{
    "result": "success",
    "swaps": [
        {
            "status": "realtime",
            "requestid": 157581336,
            "quoteid": 2717912442
        },
        {
            "status": "realtime",
            "requestid": 2050117881,
            "quoteid": 263483411
        }
    ]
}
```

5.4.4 pendingswaps

This way you can see what swaps are still pending. This uses `pending=1` arg to `swapstatus` method.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"swapstatus\",\"pending\":1}"
```

Sample Output:

```
{
    "result": "success",
    "swaps": [
        {
            "expiration": 1512738286,
```

(continues on next page)

(continued from previous page)

```
            "tradeid": 4038861795,
            "requestid": 1887485571,
            "quoteid": 2755902417,
            "iambob": 0,
            "Bgui": "",
            "Agui": "nogui",
            "gui": "nogui",
            "bob": "BTC",
            "srcamount": 0.02555775,
            "bobtxfee": 0.00212221,
            "alice": "KMD",
            "destamount": 95,
            "alicetxfee": 0.0001,
            "aliceid": "9807693779256541185",
            "sentflags": [
                "bobspend",
                "bobpayment",
                "alicepayment",
                "bobdeposit",
                "myfee",
                "bobrefund"
            ],
            "values": [
                0.02343554,
                0,
                0.02555775,
                95.0002,
                0.02875246,
                0,
                0.12226512,
                0.0279102,
                0,
                0,
                0
            ],
            "result": "success",
            "status": "finished",
            "finishtime": 1512839121,
            "bobdeposit":
↪"ec1e73395a7bd855bcf3afe58b0b1d478532ef398f2df68d2d10424e9d36743f
↪",
            "alicepayment":
↪"d3edb95bb9f3bd898f8b655398c5ab0ae86bc90d14176068096b6111eb201a05
↪",
            "bobpayment":
↪"9ff00880871d1474474378a4b9fc445ec13367be51a7b797a3cc76c1c4da52b6
↪",
            "paymentspent":
↪"0000000000000000000000000000000000000000000000000000000000000000
↪",
```

(continues on next page)

(continued from previous page)

```
            "Apaymentspent":
↪"1fd16aaf9a41e44efe30bf01087470b9db3cf287b0b7a2634b25b97601e2acda
↪",
            "depositspent":
↪"27d774b0765eee97f7423785a4c80d3cea36adaccb5fa9f4fa5fe0e8fa68bfbe
↪",
            "method": "tradestatus"
        }
    ]
}
```

5.4.5 coinswaps

This method uses `swapstatus` for a spcific coin.

Sample File Content:

```
#!/bin/bash
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"swapstatus\",\"coin\":\"CHIPS\"}"
```

5.4.6 baserelswaps

This method uses the `swapstatus` API and displays base/rel swapstatus.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"swapstatus\",\"base\":\"MNZ\",\"rel\":\
↪"KMD\"}"
```

5.4.7 `swapstatus(requestid, quoteid, pending=0)`

You can get more information about the swaps by quering with `requestid` and `quoteid`. This API helps query about stuck swap and helps them unstuck. The sample file looks like below. `pending=0` is optional.

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"swapstatus\",\"requestid\":157581336,\
↪"quoteid\":2717912442}"
```

The sample result is below:

```
{
    "requestid": 157581336,
    "quoteid": 2717912442,
    "iambob": 0,
    "bob": "REVS",
    "srcamount": 1.79816276,
    "bobtxfee": 0.0001,
    "alice": "KMD",
    "destamount": 2,
    "alicetxfee": 0.0001,
    "sentflags": [
        "alicespend",
        "bobpayment",
        "alicepayment",
        "bobdeposit",
        "myfee"
    ],
    "values": [
        1.79806276,
        0,
        1.79816276,
        2.0001,
        2.0229331,
        0,
        0.002674,
        0,
        0,
        0,
        0
    ],
    "result": "success",
    "status": "finished",
    "bobdeposit":
↪"6f36a465184f9c2a377dc7a338ff24c1096c97a7cc13aa3a8e3cf2bd46d88c03
↪",
    "alicepayment":
↪"7221eb0ac7cb9985cd580d82310a1f95b4e47617597c1b747d7840e776d73a93
↪",
    "bobpayment":
↪"7a05b28041b112178ecf5d532a18ff8e75d280fec8a7bc4511fce0215a3c66d3
↪",
    "paymentspent":
↪"9a541250257346de1b7d5b4ca11814b2df7098b750af23d0fad5b245c5435110
↪",
    "Apaymentspent":
↪"0000000000000000000000000000000000000000000000000000000000000000
↪",
    "depositspent":
↪"0000000000000000000000000000000000000000000000000000000000000000"
}
```

5.4.8 recentswaps

This api returns up to "limit" in reverse order the swaps on the node and the current pending swap (if any) so you can see what trade might happen.

'limit' = the number of latest swaps to be visible in output.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"recentswaps\",\"limit\":1}"
```

Sample Output:

```
{
    "result": "success",
    "swaps": [
        [
            2779577435,
            2461904533
        ],
        [
            932852928,
            3189912026
        ]
    ],
    "pending": {
        "expiration": 1508343494,
        "timeleft": 5,
        "base": "",
        "basevalue": 0,
        "rel": "",
        "relvalue": 0
    }
}
```

5.5 TradeBots

5.5.1 bot_buy

Starts a auto tradebot to do the trading for you. `maxprice` is the highest price you want to pay for 1 base, `relvolume` is the total amount of coins you want to spend.

Sample file contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"bot_buy\",\"base\":\"MNZ\",\"rel\":\"KMD\
↪",\"maxprice\":3,\"relvolume\":10.0}"
```

Sample output:

```
{
    "result": "success",
    "name": "buy_MNZ_KMD.1509539940",
    "botid": 3601212692,
    "started": 1509539940,
    "action": "buy",
    "base": "MNZ",
    "rel": "KMD",
    "maxprice": 3,
    "totalrelvolume": 10,
    "totalbasevolume": 3.33333333,
    "trades": []
}
```

5.5.2 bot_list

This script will display a list of all `botids` that you used.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"bot_list\"}"
```

Sample Output:

```
[
  2833291137,
  3601212692,
  2712409213
]
```

5.5.3 bot_pause

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"bot_pause\",\"botid\":$1}"
```

Sample usage: `./bot_resume 3601212692` # here **3601212692** is the `botid` from `./bot_buy` or `./bot_sell` script output.

Sample Output:

```
{
        "result":"success"
}
```

5.5.4 bot_resume

Resumes a paused tradebot.

Sample file contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"bot_resume\",\"botid\":$1}"
```

Sample Usage: `./bot_resume 3601212692` # here **3601212692** is the `botid` from `./bot_buy` or `./bot_sell` script output.

Sample Output:

```
{
    "result": "success"
}
```

5.5.5 bot_sell

Place a sell order using tradebots. `minprice` is the lowest you will sell for 1 coin, `basevolume` is the total amount of coins you want to sell.

Sample file contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"bot_sell\",\"base\":\"REVS\",\"rel\":\
↪"KMD\",\"minprice\":2,\"basevolume\":5.0}"
```

Sample Output:

```
{
    "result": "success",
    "name": "sell_MNZ_KMD.1509549405",
    "botid": 2247122510,
    "started": 1509549405,
    "action": "sell",
    "base": "MNZ",
    "rel": "KMD",
    "minprice": 2,
    "totalbasevolume": 5,
    "totalrelvolume": 10,
    "trades": []
}
```

5.5.6 bot_settings

`./bot_settings` allows you to change price or volume after starting the `bot_buy` or `bot_sell`.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"bot_settings\",\"botid\":$1,\"newprice\":
↪$1,\"newvolume\":$2}"
```

Sample Usage:

`./bot_settings 2247122510 3 6` # here **2247122510** is `botid` and 3 is the `new price`, 6 is the `new volume`

Sample Outputs:

```
{
    "result": "success",
    "name": "sell_MNZ_KMD.1509549405",
    "botid": 2247122510,
    "started": 1509549405,
    "paused": 1509549517,
    "action": "sell",
    "base": "MNZ",
    "rel": "KMD",
    "minprice": 2247122510,
    "totalbasevolume": 0,
    "totalrelvolume": 3,
    "aveprice": 2,
    "volume": 4,
    "trades": [
        {},
        {},
        {},
        {},
        {},
        {},
        {},
        {},
        {},
        {}
    ],
    "completed": 4,
    "percentage": 149808167333.3333,
    "pending": 6,
    "pendingprice": 2,
    "pendingvolume": 6
}
```

5.5.7 bot_status

Tradebot status. Use `botid` after the `./bot_status` script.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"bot_status\",\"botid\":$1}"
```

Sample Usage: `./bot_status 3601212692` # here **3601212692** is the `botid` from `./bot_buy` or `./bot_sell` script output.

Sample Output:

```
{
    "result": "success",
    "name": "buy_MNZ_KMD.1509542273",
    "botid": 2384827585,
    "started": 1509542273,
    "paused": 1509542452,
    "action": "buy",
    "base": "MNZ",
    "rel": "KMD",
    "maxprice": 0.21,
    "totalrelvolume": 0.1,
    "totalbasevolume": 0.47619048,
    "aveprice": 0.20486144,
    "volume": 0.0672,
    "trades": [
        {},
        {},
        {},
        {},
        {},
        {},
        {},
        {},
        {},
        {},
        {}
    ],
    "completed": 7,
    "percentage": 67.2,
    "pending": 4,
    "pendingprice": 0.20633316,
    "pendingvolume": 0.0387
}
```

5.5.8 bot_statuslist

Displays the status of all bots.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"bot_statuslist\"}"
```

(continues on next page)

(continued from previous page)

5.5.9 bot_stop

To stop the tradebots. Use `botid` from your buy or sell order.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"bot_stop\",\"botid\":$1}"
```

Sample usage: `./bot_stop 3601212692` # here **3601212692** is the `botid` from `./bot_buy` or `./bot_sell` script output.

Sample Output:

```
{
    "result": "success"
}
```

5.6 Coin Features

5.6.1 balance

This will display the balance of a specific smartaddress for barterDEX trading. The idea of the barterDEX balance are, `utxo`. That are "hot" / "live" ready for trading. It is the analogue of funds in an exchange account. This will not show your full wallet balance, and you wouldn't want the exchange account balance to show the amount of all your coins, including the ones not deposited in the exchange. Edit the file with coin `name` and `smartaddress` and it will display balance of your smartaddress(s).

KMD balance will display `zcredits` as well.

Sample File Contents:

```
#!/bin/bash
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"balance\",\"coin\":\"KMD\",\"address\":\
↪"RHV2As4rox97BuE3LK96vMeNY8VsGRTmBj\"}"
```

Sample Output:

```
{
    "result": "success",
    "coin": "KMD",
```

(continues on next page)

(continued from previous page)

```
    "address": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
    "balance": 360.33996021,
    "zcredits": 0,
    "zdebits": {
        "swaps": [],
        "pendingswaps": 0
    }
} {
    "result": "success",
    "coin": "REVS",
    "address": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
    "balance": 39.10219798
} {
    "result": "success",
    "coin": "BTC",
    "address": "126nKAnGxWtsfPoxGh1snvbtLitkstZbvV",
    "balance": 0.00400000
}
```

5.6.2 balances

This method will display balance of all active coins in your wallet. No need to use a curl script for a specific coin and address.

Sample File Content:

```
#!/bin/bash
source userpass
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"balances\"}"
```

Sample Output:

```
[
    {
        "coin": "BTC",
        "balance": 0.00400000
    },
    {
        "coin": "KMD",
        "balance": 360.33996021
    },
    {
        "coin": "MNZ",
        "balance": 55.70756040
    },
    {
        "coin": "REVS",
```

(continues on next page)

(continued from previous page)

```
        "balance": 39.10219798
    },
    {
        "coin": "JUMBLR",
        "balance": 1.12038776
    },
    {
        "coin": "SUPERNET",
        "balance": 0.91430900
    },
    {
        "coin": "PANGEA",
        "balance": 1
    },
    {
        "coin": "DEX",
        "balance": 1
    },
    {
        "coin": "BET",
        "balance": 1
    },
    {
        "coin": "CRYPTO",
        "balance": 1
    },
    {
        "coin": "HODL",
        "balance": 1
    },
    {
        "coin": "BOTS",
        "balance": 1
    },
    {
        "coin": "COQUI",
        "balance": 10.85596031
    }
]
```

5.6.3 calcaddress

Useful to get your private key for specific seed passphrase for any supported coin or asset except ETH & ERC20 tokens for now. Edit the passphrase section and change the word `default` with your own seed passphrase and coin name. We used Digibyte (DGB) as example.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"calcaddress\",\"passphrase\":\"default\",
↪\"coin\":\"DGB\"}"
```

Sample Output: This output is based on the passphrase `default`. Change that to your seed passphrase to get your desired private key.

```
{
    "passphrase": "default",
    "pubsecp":
↪"03562035ac51f940b7f6e1e390df91fb0fe7f59f99acf8f6f389d9472a14797b5f
↪",
    "coinaddr": "DKRQHUtFhFfViZPCcCut3uB45cATTXXBMh",
    "p2shaddr": "3FyKfmS3wk5bGiu2ziauvmNPLzisePpY3W",
    "privkey":
↪"30a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0684f
↪",
    "wif": "KxrJKHQ7phDnVnsbs2XEZUeWxjH3zSG6SzftyQGfojEG7ZkU9Zsn"
}
```

5.6.4 fundvalue

`fundvalue` will process `"holdings":[]` array and also do an internal balances api call if `address` is specified. This will only work for watchonly addresses that have been rescanned and with active coins. In order for external addresses to work, you need to have `importaddress` and `rescan` done, for obvious reasons, in native mode. SPV mode should just work without that.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"fundvalue\",\"address\":\
↪"RRyyejME7LRTuvdziWsXkAbSW1fdiohGwK\"}"
```

Sample Output:

```
{
    "holdings": [
        {
            "coin": "KMD",
            "KMD": 6000000
        },
        {
            "coin": "REVS",
            "error": "no price source"
        },
        {
            "coin": "JUMBLR",
```

(continues on next page)

(continued from previous page)

```
                "KMD": 4177502.22193679
            },
            {
                "coin": "SUPERNET",
                "error": "no price source"
            },
            {
                "coin": "PANGEA",
                "error": "no price source"
            },
            {
                "coin": "DEX",
                "error": "no price source"
            },
            {
                "coin": "BET",
                "error": "no price source"
            },
            {
                "coin": "CRYPTO",
                "error": "no price source"
            },
            {
                "coin": "HODL",
                "error": "no price source"
            },
            {
                "coin": "MSHARK",
                "error": "no price source"
            },
            {
                "coin": "BOTS",
                "error": "no price source"
            }
        ],
        "fundvalue": 10177502.22193679
}
```

5.6.5 getrawtransaction

barterDEX now has `getrawtransaction` API for any installed coin and will use Electrum

(continued from previous page)

5.6.6 inuse

This API will return you all the UTXOs that are locked for swaps. The UTXOs that are locked for swaps are not spendable until they become free. The lock is basically for about 2 minutes, after that the UTXOs becomes free.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"inuse\"}"
```

Sample Output:

```
[
    {
        "expiration": 1520277125,
        "txid":
↪"74a9f85dc372c3941bdfa7bc517f209db11c2fbdf7c83df3f4c8ccf15afa9e75
↪",
        "vout": 1
    },
    {
        "expiration": 1520277125,
        "txid":
↪"46f021a999b5f62fc42afeb431d09c85a2f2bc113aa23b5142cafb868a877bd0
↪",
        "vout": 1
    }
]
```

```
[
    {
        "account": "",
        "address": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
        "category": "receive",
        "amount": 0.26042583,
        "vout": 0,
        "confirmations": 8904,
        "blockhash":
↪"00009eeaa7458a5d781830c166158481fda6ae039450607f90e239d839694050
↪",
        "blockindex": 2,
        "blocktime": 1521983789,
        "txid":
↪"2ff201697e8486865638d35f4de796ce025ec72734c4053a6b835dfb50340ca9
↪",
        "walletconflicts": [],
        "time": 1521983789,
        "timereceived": 1522155873,
        "vjoinsplit": [],
        "size": 300
    },
    {
        "account": "",
        "address": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
        "category": "receive",
        "amount": 0.00203615,
        "vout": 1,
        "confirmations": 5955,
        "blockhash":
↪"00000e1d26d1e13cb9c9456f2f2112fd2c3079fbdf66a88dbbf96db6114aecfe
↪",
        "blockindex": 1,
        "blocktime": 1522161804,
        "txid":
↪"2235aa0b2beed7b2a79619df044912af3baec026cda6a4eb00c9de5b6a1f6788
↪",
        "walletconflicts": [],
        "time": 1522161801,
        "timereceived": 1522161801,
        "vjoinsplit": [],
        "size": 223
    },
    {
        "account": "",
        "address": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
        "category": "send",
        "amount": -0.00203615,
        "vout": 1,
        "fee": -1e-05,
```

(continues on next page)

(continued from previous page)

```
        "confirmations": 5955,
        "blockhash":
↪"00000e1d26d1e13cb9c9456f2f2112fd2c3079fbdf66a88dbbf96db6114aecfe
↪",
        "blockindex": 1,
        "blocktime": 1522161804,
        "txid":
↪"2235aa0b2beed7b2a79619df044912af3baec026cda6a4eb00c9de5b6a1f6788
↪",
        "walletconflicts": [],
        "time": 1522161801,
        "timereceived": 1522161801,
        "vjoinsplit": [],
        "size": 223
    },
    {
        "involvesWatchonly": true,
        "account": "",
        "address": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
        "category": "receive",
        "amount": 14.4499848,
        "vout": 0,
        "confirmations": 4278,
        "blockhash":
↪"0000f759c96b9c866e6417bb3c653e323feea999ce1dabcc27c007f72543f14c
↪",
        "blockindex": 1,
        "blocktime": 1522261902,
        "txid":
↪"6b2b506cffe3423e25d3519c64a5279213993e0d235ff89cbb3bf53be7a727fb
↪",
        "walletconflicts": [],
        "time": 1522261807,
        "timereceived": 1522261807,
        "vjoinsplit": [],
        "size": 300
    },
    {
        "involvesWatchonly": true,
        "account": "",
        "address": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
        "category": "send",
        "amount": -14.4499848,
        "vout": 0,
        "fee": -1e-05,
        "confirmations": 4278,
        "blockhash":
↪"0000f759c96b9c866e6417bb3c653e323feea999ce1dabcc27c007f72543f14c
↪",
```

(continues on next page)

(continued from previous page)

```
        "blockindex": 1,
        "blocktime": 1522261902,
        "txid":
↪"6b2b506cffe3423e25d3519c64a5279213993e0d235ff89cbb3bf53be7a727fb
↪",
        "walletconflicts": [],
        "time": 1522261807,
        "timereceived": 1522261807,
        "vjoinsplit": [],
        "size": 300
    }
]
```

Sample output using SPV mode:

```
[
    {
        "tx_hash":
↪"7c536b8f3d1e33f110ae62399f181b4c5af9a16938b1ecef59f48c226dd2a9be
↪",
        "height": 8285
    },
    {
        "tx_hash":
↪"f3169db98c3b7263130d8d0d91aa17df361b76737f8df8363e3b4caeb1afaa55
↪",
        "height": 8288
    },
    {
        "tx_hash":
↪"24874d70f3cfa5ae3a1c31916af7bdc6cb6d6fe5d428a12873c373db703bd101
↪",
        "height": 8833
    },
    {
        "tx_hash":
↪"9281bcfc9c3262cb3e6d8440fd0250357a8660a24aa22abeca8ce7512c0cf111
↪",
        "height": 9012
    },
    {
        "tx_hash":
↪"c3561504f2ce0af8f5c63fa01424d6acb5576caddfa543dc3e16301d51f744f6
↪",
        "height": 9013
    },
    {
        "tx_hash":
↪"dce11261eaae611347a5967ea727f0bc0a3ee87e11250ad2ef0758a999d7e920
↪",
```

(continues on next page)

(continued from previous page)

```
        "height": 9602
    },
    {
        "tx_hash":
↪"afc2fd0991bdef89da2cd1018c86ab56a5989c6f2b1d4e40861e88bc6125cbec
↪",
        "height": 10076
    }
]
```

5.6.8 listunspent

barterDEX uses `utxo` pairs instead of total balance to do the trading. For this the balance has be broken down and fund the address using 1x, 1.2x & 0.1x ratio. It is due to the nature of atomic swap protocol that is bartering specific utxo and the need for two utxo per atomic swap. **Analogy: If you need a quarter and a dime, exactly to make a purchase, is it possible to use just a single coin? barterDEX is swapping PAIRS (as in two) utxos.** `./listunspent` Gives similar output like your coin wallet. It will list all the unspent transactions (`utxo`) you have in a given coin wallet smartaddress.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"listunspent\",\"coin\":\"CHIPS\",\
↪"address\":\"RDoMmoCM5AcEH6Yf5vqKbqRjD1fLcQHuWb\"}"
```

Sample Output:

```
[
    {
        "tx_hash":
↪"873e12d89eb9a25c1c40f0301b77414da0567982e3a06ce55a3501ff726bd253
↪",
        "tx_pos": 1,
        "height": 534479,
        "value": "1251800000"
    },
    {
        "tx_hash":
↪"8a62bb10f700c16629a52ad0dc8c730490de3038e30af82b1a706c91f337d968
↪",
        "tx_pos": 1,
        "height": 535408,
        "value": "11000000"
    },
    {
        "tx_hash":
↪"8848abd97daca43c702dbb176dac50d85261449c808f19324f5be1b4a495ad69
↪",
```

(continues on next page)

(continued from previous page)

```
            "tx_pos": 1,
            "height": 535379,
            "value": "11000000"
        },
        {
            "tx_hash":
↪"79684e9c19c996292ad4a25c9dc05c89a3a9243af1b83727362bb2e2f92e08f6
↪",
            "tx_pos": 1,
            "height": 534027,
            "value": "1012700000"
        }
]
```

5.6.9 secretaddresses

It will display secret 10 addresses with their private keys for a given `pubtype`. You can change the `num`, `pubtype` etc by editing the script.

Sample File Contents:

```
echo "usage: ./secretaddresses 'passphrase'"

curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"secretaddresses\",\"num\":16,\
↪"passphrase\":\"$1\"}"
```

Sample Output:

```
usage: ./secretaddresses 'passphrase'
{"R9oeSNXqR499PFz2BgUeZJvSsp331TFDYi":"UxKrG3xxxxxx",
"RX7sUCvxV45LmrmVRvkbm8kUedFqhXgqtt":"UuNeHYexxxxxx",
"RRKf3RnmbNDnCfC6P7HHYNBJuRrVxEm7Db":"UsXUNtmxxxxxx",
"RSo6exWzMbnGnwM4b3WcT1MvxvnRuRSURP":"UsFuDaWxxxxxx",
"RRve3cdBJPF2dcT6uD8gT2Bfhg7ZU9krFU":"Uw1Xbcoxxxxxx",
"REkV8JMBuLFTjYH93QfJ4UpmoLw85LomP4":"UpYxoFMxxxxxx",
"RKJ6vpCTYjDCoyrLyuxJRvjiraM5ZEwos5":"Urin6VUxxxxxx",
"RDHo38pWxDEvqx99C2aJhprHQwf9bYoKRE":"UuNsFyBxxxxxx",
"RLGRaoQ7FCu8dk653CMR4nLiRnfYZiXb7D":"UvT2utaxxxxxx",
"RJABFzJzBNoPUGZrMBcBfL5UqYjY8hptJU":"UsoAJqRxxxxxx",
"RN2TRHQBFWMB9G1MkHKdYbg4oU7ToxRTdB":"UqNiGgAxxxxxx",
"RJYuYeXAnWoVPbgHpc1rtxkoBWFCtzib1E":"Uqeaxmzxxxxxx",
"RXkZYAD13G89foJbN7uHpD2A2oZJaDTMzr":"UqejSkPxxxxxx",
"RUfuMah7a9QXar5XcJpage4S7jtoiL99Kv":"Uq7e4Hdxxxxxx",
"RTS385Ny2feDBgLGzrAPRw2Q7u5jpv9jZs":"UvUcSEPxxxxxx",
"RP7haeUQv56FW9X6W1rRbhbqvXPyvX23Fi":"UpqRcKMxxxxxx"}
```

5.6.10 sendrawtransaction

This method will broadcast a raw transaction over the network. Uses tx-hex from `./withdraw` API output.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"sendrawtransaction\",\"coin\":\"KMD\",\
↪"signedtx\":\"$1\"}"
```

5.6.11 supernet

`supernet` script uses the `balances` API and returs all SuperNET aseet balances along with KMD for a specific address. In order for external addresses to work, you need to have `importaddress` and `rescan` done, for obvious reasons, in native mode. SPV mdoe just works without that.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"balances\",\"address\":\
↪"RRyyejME7LRTuvdziWsXkAbSW1fdiohGwK\"}"
```

Sample Output:

```
[
    {
        "coin": "KMD",
        "balance": 6000000
    },
    {
        "coin": "REVS",
        "balance": 6335.95500000
    },
    {
        "coin": "JUMBLR",
        "balance": 239022.38900000
    },
    {
        "coin": "SUPERNET",
        "balance": 6335.95500000
    },
    {
        "coin": "PANGEA",
        "balance": 602828.14640000
    },
    {
        "coin": "DEX",
```

(continues on next page)

(continued from previous page)

```
        "balance": 257733
    },
    {
        "coin": "BET",
        "balance": 2688
    },
    {
        "coin": "CRYPTO",
        "balance": 320754.73700000
    },
    {
        "coin": "HODL",
        "balance": 70000
    },
    {
        "coin": "MSHARK",
        "balance": 385403.90000000
    },
    {
        "coin": "BOTS",
        "balance": 288157
    }
]
```

5.6.12 timelock and unlockedspend

BarterDEX now has a coin agnostic timelock/unlock mechanism. Of course the coin will need to support CLTV or it wont work. The timelock will work partially, but the unlockspend will NOT. Test with small amount. Don't confuse these APIs with 0conf deposit and claim.

Sample file content:

`timelock`

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"timelock\",\"coin\":\"KMD\",\"duration\
↪":1000,\"amount\":1}"
```

`unlockedspend`

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"unlockedspend\",\"coin\":\"KMD\",\"txid\
↪":\
↪"e858e382a816b4cab22e3fd3e29901c7ef497cd1fdad7683314cc9187eca34fd\
↪"}"
```

`timelock` API is pretty simple, just specify coin, duration in seconds and the amount. An extra txfee will be added to amount so the unlockspend can send the specified amount. there is an optional `destaddr` field that will allow you to send to any address. Like `withdraw`, timelock will not actually broadcast the tx, so you need to use sendrawtransaction to actually broadcast it.

Once its broadcast, and the time lock expires, you can do an unlockedspend using just the coin and the txid. The requirement is that it must be sent to your smartaddress that you are logged into and it will only just unlock and send to yourself.

5.6.13 withdraw

To withdraw, you just need to specify the array of outputs and the coin. It returns the rawtx, the signed transaction in hex the txid and if it was complete or not. All inputs are assumed to be standard pay to pubkeyhash, having other non-standard utxo will make it create invalid rawtx. Withdrawing ETH/ERC20 tokens works using different parameters (please see *eth_withdraw*,).

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"withdraw\",\"coin\":\"KMD\",\"outputs\":[
↪{\"RUgW6fLfVsLJ87Ng4zJTqNedJSKYQ9ToAf\":0.001}, {\
↪"RUgW6fLfVsLJ87Ng4zJTqNedJSKYQ9ToAf\":0.002}],\"broadcast\":1}"
```

Sample Output:

```
{
    "method": "withdraw",
    "coin": "KMD",
    "outputs": [
        {
            "RUgW6fLfVsLJ87Ng4zJTqNedJSKYQ9ToAf": 0.00100000
        },
        {
            "RUgW6fLfVsLJ87Ng4zJTqNedJSKYQ9ToAf": 0.00200000
        }
    ]
} ->␣
↪f6b04d157c007bf688822a73c754934f53cf28516f64dc64372fa397789e9ce1
    {
    "rawtx":
↪"0100000001b5384fc2f0b737a17239eaf2172bf24c4f410f720e672c3f56a9b5a8e0b33a91010
↪",
    "hex":
↪"0100000001b5384fc2f0b737a17239eaf2172bf24c4f410f720e672c3f56a9b5a8e0b33a91010
↪",
    "txid":
↪"f6b04d157c007bf688822a73c754934f53cf28516f64dc64372fa397789e9ce1
↪",
    "complete": true
}
```

If the last output is doing a withdraw to the smartaddress, it is combined with change. Here are two sample console output below. The first one is withdrawing 10 KMD to the same smartaddress. Second one is 10 & 10.0001 KMD withdraw to the same smartaddress. The last output is increased by 0.0001 & change combined in both example.

```
vout.0 10.00010000 -> total 10.00020000
dustcombine.0 numpre.0 min0.(nil) min1.(nil) numutxos.172 amount 10.
↪00020000
minutes.381 tiptime.1514745296 locktime.1514722428
46bbfbe93ebfc0a9262143193fe50d5c9ce5c2739da768ce8b5ec919ffaa570f/0␣
↪40.01342800 interest 0.00122360 -> sum 0.00122360
numunspents.171 vini.0 value 40.01342800, total 40.01342800 remains␣
↪-30.01322800 interest 0.00122360 sum 0.00122360␣
↪46bbfbe93ebfc0a9262143193fe50d5c9ce5c2739da768ce8b5ec919ffaa570f/
↪v0
set inuse until 1514746673 lag.600 for␣
↪46bbfbe93ebfc0a9262143193fe50d5c9ce5c2739da768ce8b5ec919ffaa570f/
↪v0
change 30.01445160 = total 40.01465160 - amount 10.00020000, adjust␣
↪0.00000000 numvouts.1
combine last vout 10.00010000 with change 30.01445160
LP_withdraw.KMD {"method":"withdraw","coin":"KMD","outputs":[{
↪"RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ":10.00010000}]} ->␣
↪d3f32d814e20edcbc0c654eec79d0399b38b907ecc8cb9418da17a543d299e17
```

```
vout.0 10.00000000 -> total 10.00010000
vout.1 10.00010000 -> total 20.00020000
dustcombine.0 numpre.0 min0.(nil) min1.(nil) numutxos.172 amount 20.
↪00020000
numunspents.171 vini.0 value 40.01455160, total 40.01455160 remains␣
↪-20.01435160 interest 0.00000000 sum 0.00000000␣
↪d3f32d814e20edcbc0c654eec79d0399b38b907ecc8cb9418da17a543d299e17/
↪v0
set inuse until 1514747480 lag.600 for␣
↪d3f32d814e20edcbc0c654eec79d0399b38b907ecc8cb9418da17a543d299e17/
↪v0
change 20.01435160 = total 40.01455160 - amount 20.00020000, adjust␣
↪0.00000000 numvouts.2
combine last vout 10.00010000 with change 20.01435160
LP_withdraw.KMD {"method":"withdraw","coin":"KMD","outputs":[{
↪"RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ":10}, {
↪"RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ":10.00010000}]} ->␣
↪f24361b13b7f0188ea422572b882c0df50bef2a33bf8aed7bf0776946332cece
```

5.6.14 eth_withdraw

This method is required to withdraw ETH/ERC20 coin/tokens. Similar functionality to `withdraw` for Bitcoin protocol based coins.

```
curl --url "http://127.0.0.1:7783 " --data "{\"userpass\":\"
↪$userpass\",\"method\":\"eth_withdraw\",\"coin\":\"DEC8\",\"to\":\
↪"0xbAB36286672fbdc7B250804bf6D14Be0dF69fa29\",\"amount\":1000}"
```

5.6.15 opreturn

`OP_RETURN` is just built into the `withdraw` API. If an `"opreturn":"<hex>"` is defined, it will create an OP_RETURN with the binary of the hex. Without `outputs` value, it will return `{"error":"withdraw needs to have outputs"}`. There is encryption to the opreturn mode of withdraw. Basically if you specify a non-null passphrase, it encrypts with that passphrase and you will need that passphrase to decode it. The second parameter in the first example script is the passphrase. So, if you don't specify it, it is unencrypted.

3 different sample scripts:

```
curl -X POST --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"withdraw\",\"coin\":\"COQUI\",\
↪"passphrase\":\"$2\", \"opreturn\":\"$(echo -n "$1" | od -A n -t␣
↪x1 | perl -pe 's/\W+//g')\"}"
```

```
curl -X POST --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"withdraw\",\"coin\":\"COQUI\",\"outputs\
↪":[{\"RGCVXhUxepCxh77thy7XgMrQ5XU9u7SWSR\":0.01}], \"opreturn\":\"
↪$(echo -n "$1" | od -A n -t x1 | perl -pe 's/\W+//g')\"}"
```

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"withdraw\",\"coin\":\"KMD\",\"outputs\":[
↪{\"RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ\":$1}], \"opreturn\":\
↪"deadbeef\"}"
```

Sample Output:

```
{
    "rawtx":
↪"01000000029059515dd331b448b213c12e607b4f104a025f9461d1c094d98010a06406b62d010
↪",
    "hex":
↪"01000000029059515dd331b448b213c12e607b4f104a025f9461d1c094d98010a06406b62d010
↪",
    "tx": {
        "version": 1,
        "locktime": 1520462106,
        "vin": [
            {
                "txid":
↪"2db60664a01080d994c0d161945f024a104f7b602ec113b248b431d35d515990
↪",
                "vout": 1,
```

(continues on next page)

(continued from previous page)

```
                "scriptPubKey": {
                    "hex":
↪"76a9140c1007fc1f406a0a519886c0e59327e9c43a634088ac"
                }
            },
            {
                "txid":
↪"648b5c66d5571ca26d2055ed21117baeef2caf643ce0d79cf68d21137afc4368
↪",
                "vout": 1,
                "scriptPubKey": {
                    "hex":
↪"76a9140c1007fc1f406a0a519886c0e59327e9c43a634088ac"
                }
            }
        ],
        "vout": [
            {
                "satoshis": "1530966",
                "scriptPubKey": {
                    "hex":
↪"76a9140c1007fc1f406a0a519886c0e59327e9c43a634088ac"
                }
            },
            {
                "satoshis": "0",
                "scriptPubKey": {
                    "hex": "6a04deadbeef"
                }
            }
        ]
    },
    "txid":
↪"4e2a338232d13643cdbd6fb0c47d649b9c1842284a78ab6ca9cf03088f3dc0cc
↪",
    "complete": true
}
```

5.6.16 opreturndecrypt

This method will decrypt the encrypted *OP_RETURN* easily.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"opreturndecrypt\",\"coin\":\"KMD\",\
↪"txid\":\
↪"06e59c6e9217ef1e526a1419e231e5f927689765feab9cd48167e089301f6fc2\
↪",\"passphrase\":\"test\"}"
```

(continues on next page)

(continued from previous page)

```
```

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"opreturndecrypt\",\"coin\":\"KMD\",\
↪"txid\":\
↪"34bf4bc2247e43c0c6a2223b79d3f1a3e9962102fb4a563612e5bbb91fb85348\
↪",\"passphrase\":\"test\"}"
```

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"opreturndecrypt\",\"coin\":\"KMD\",\
↪"txid\":\
↪"a9e94e6890d6998b3c2b89685e8c7c2b9e380a53fb813192701d0966480807eb\
↪",\"passphrase\":\"test\"}"
```

Sample Output:

```
{
    "coin": "KMD",
    "opreturntxid":
↪"06e59c6e9217ef1e526a1419e231e5f927689765feab9cd48167e089301f6fc2
↪",
    "opreturn":
↪"6a4c4c4c0058501c2c290f7241932a8b4614b844bae28cc3117eeb79d5c0c9000000000000000
↪",
    "result": "success",
    "decrypted": "68656c6c6f2c20776f726c6474657374",
    "original": "hello, worldtest"
}
```

```
{
    "coin": "KMD",
    "opreturntxid":
↪"34bf4bc2247e43c0c6a2223b79d3f1a3e9962102fb4a563612e5bbb91fb85348
↪",
    "opreturn":
↪"6a4c4d4d00215bb9fee0d7685c5fcf913f4da4fab726104b1b242d5226939b000000000000000
↪",
    "result": "success",
    "decrypted": "68656c6c6f2c20776f726c647465737431",
    "original": "hello, worldtest1"
}
```

```
{
    "coin": "KMD",
    "opreturntxid":
↪"a9e94e6890d6998b3c2b89685e8c7c2b9e380a53fb813192701d0966480807eb
↪",
    "opreturn":
↪"6a4b4b0057a7b6c2ac0354f967534060bc3719a687bedc3b4741c3c64b7d00000000000000000
↪",
```

(continues on next page)

(continued from previous page)

```
    "result": "success",
    "decrypted": "68656c6c6f2c20776f726c64746573",
    "original": "hello, worldtes"
}
```

All three recreated the original string passed into the *opreturn* script. This functionality should allow apps to easily created and retrieve encrypted `OP_RETURN`.

5.7 Statistics

5.7.1 guistats

`guistats` method is for GUI apps to display statistics, based on the `statsdisp` API.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"statsdisp\",\"starttime\":0,\"endtime\
↪":0,\"gui\":\"SimpleUI\"}"
```

5.7.2 pricearray

pricearray will display statistical price data based on timescale.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"pricearray\",\"base\":\"REVS\",\"rel\":\
↪"KMD\",\"timescale\":300}"
```

This will output the data in an order of `[timestamp, avebid, aveask, highbid, lowask]`

Sample output:

```
[
 [1510603500, 1.40944454, 1.40944453, 3.13224635, 1.07003092],
 [1510603800, 1.50064727, 1.50064726, 3.13224635, 1.07003092],
 [1510604100, 1.50064727, 1.50064726, 3.13224635, 1.07003092],
 [1510607100, 1.44868382, 1.44868381, 3.13224635, 1.07003092],
 [1510607400, 1.50064727, 1.50064726, 3.13224635, 1.07003092],
 [1510607700, 1.52877070, 1.52877069, 3.13224635, 1.07003092],
 [1510608000, 1.55019807, 1.55019806, 3.13224635, 1.09860370],
 [1510608300, 1.57205536, 1.57205536, 3.13224635, 1.09860370],
 [1510608600, 1.30344136, 1.30344135, 1.76975636, 1.17393764],
 [1510611600, 1.21488583, 1.21488582, 1.25583401, 1.17393764],
```

(continues on next page)

(continued from previous page)

```
 [1510611900, 1.25661049, 1.25661048, 1.34097978, 1.17393764]
]
```

5.7.3 statsdisp

`statsdisp` processes the `stats.log` file on your node. It will display the swaps as best as it can figure out the unique number of swaps, you can set specific `starttime` and `endtime`. If not specified, will display all. If both are the same time from the future, it will display only the pending stats.

This is a very good feature as it can identify any node that is trying to do too many trades at once

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"statsdisp\",\"starttime\":0,\"endtime\
↪":0}"
```

Sample Output:

```
{
    "result": "success",
    "newlines": 0,
    "request": 0,
    "reserved": 0,
    "connect": 0,
    "connected": 0,
    "duplicates": 0,
    "parse_errors": 0,
    "uniques": 0,
    "tradestatus": 0,
    "unknown": 0
}
```

5.7.4 ticker

`ticker` API returns trades for the previous 24hours. You can set `base/rel` optionally. Most useful for price and marketcap statistics.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"ticker\",\"base\":\"REVS\",\"rel\":\"KMD\
↪"}"
```

Sample Output:

```
[
    {
        "timestamp": 1513174486,
        "KMD": 4.02827942,
        "CHIPS": 80.00010000,
        "price": 19.85962036
    },
    {
        "timestamp": 1513168630,
        "KMD": 0.07448089,
        "MNZ": 0.25609997,
        "price": 3.43846549
    },
    {
        "timestamp": 1513166465,
        "CRYPTO": 44.00844307,
        "KMD": 500.00010000,
        "price": 11.36145851
    },
    {
        "timestamp": 1513158008,
        "KMD": 463.91820708,
        "SUPERNET": 10.00010000,
        "price": 0.02155574
    }
]
```

5.7.5 tradesarray

`tradesarray` will display statistical trade data based on timescale in seconds.

Sample File Content:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"tradesarray\",\"base\":\"REVS\",\"rel\":\
↪"KMD\",\"timescale\":120}"
```

This will output the data in an order of `[timestamp, high, low, open, close, relvolume, basevolume, aveprice, numtrades]`

Sample output:

```
[
 [1511480400, 1.27614172, 1.27614172, 1.27614172, 1.27614172, 0.
↪04990000, 0.03910224, 1.27614172, 1],
 [1511480520, 1.97936698, 1.36595231, 1.97936698, 1.36595231, 0.
↪09980000, 0.06174137, 1.61642024, 2],
 [1511480640, 2.11837065, 2.11837065, 2.11837065, 2.11837065, 0.
↪04990000, 0.02355584, 2.11837065, 1],
```

(continues on next page)

(continued from previous page)

```
[1511481840, 1.63195519, 1.55147942, 1.63195519, 1.55147942, 0.
↪09980000, 0.06273967, 1.59070011, 2],
[1511481960, 1.82528276, 1.82528276, 1.82528276, 1.82528276, 0.
↪04990000, 0.02733823, 1.82528276, 1],
[1511482680, 1.77777461, 1.77777461, 1.77777461, 1.77777461, 0.
↪04990000, 0.02806880, 1.77777461, 1]]
```

5.8 Communication

5.8.1 deletemessages

As the name suggests this API will let you delete messages received from other nodes.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"deletemessages\",\"firsti\":0,\"num\":10}
↪"
```

Sample Output:

```
{
        "result": "success"
}
```

5.8.2 getmessages

This API will allow to retrieve messages sent from other nodes.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"getmessages\",\"firsti\":0,\"num\":10}"
```

Sample Output:

```
{
          "messages": []
}
```

5.8.3 message

The `./message` script let nodes communicate with each other with end to end encryption. It is recommended to message a node first before doing an actual trade to make sure they can communicate.

To send a message you need to have the recipients public key. The public key can be found when launching the `./client` or running any other script the first time after launching `./client`. Edit the `./message` script to write your custom message. Execute `./message <pubkey of recipient>` to send the message.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"pubkey\":\"$1\",\"method\":\"sendmessage\",\
↪"message\":\"some sort of message\"}"
```

Sample Output:

```
{
            "result": "success"
}
```

5.9 Revenue Sharing/Operations

5.9.1 dividends

This script will allow users / organizations to send dividends for any specific coins/asstets

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"dividends\",\"coin\":\"KMD\",\"height\":
↪$1,\"prefix\":\"fiat/jumblr sendtoaddress\",\"suffix\":\"\",\
↪"dividend\":50000,\"dust\":1}"
```

5.9.2 snapshot

The name says it all, snapshot will take a snap of the blockchain for a specified coin/asset on a certain block height for sending dividends or airdrop.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"snapshot\",\"coin\":\"KMD\",\"height\":
↪$1}"
```

5.9.3 snapshot_balance

This API will display the snapshot balance of a specific coin address after snapshot taken on a specific block height. You need to have jq installed (https://stedolan.github.io/jq/).

Sample File Contents:

```
ht=$1
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"snapshot_balance\",\"coin\":\"KMD\",\
↪"height\":$ht,\"addresses\":[\"RSAzPFzgTZHNcxLNLdGyVPbjbMA8PRY7Ss\
↪", \"RBgD5eMGwZppid4x7PTEC2Wg1AzvxbsQqB\"]}"
```

5.9.4 snapshot_loop

This API is used to automate snapshot process for coins/assets at given block height and certain number thereafter. You can configure the script to run on certain block height increments.

Sample File Contents:

```
ht=$1
while true
do
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"snapshot\",\"coin\":\"KMD\",\"height\":
↪$ht}"
#ht=`komodo-cli getinfo | jq .blocks`
ht=$(( $ht + 1000 ))
echo next height $ht
sleep 1
done
```

5.10 InstantDEX SWAP

Fastest swaps are here now as InstantDEX Swap. All nodes can be doing this at the same time, any `bob` to any `alice`. All you need to have is some amount of KMD deposit for dynamic trust. It does require a deposit that matches your trade size.

If you are selling 20 at at time, deposit 20. If you are selling 50 at a time, deposit 50. And, you can trade without any deposit if you want to wait for coin confirmations. Use `InstantDEX` swap, just wait for deposit coin confirmations, you can sell all. The deposit just has to cover the amount you have pending in swaps, not anything regarding your total amount.

You can see your balance using the regular `balance` API. After verification your `zcredits` will be displayed with your KMD balance like this:

```
{
    "result": "success",
    "coin": "KMD",
    "address": "RANyPgfZZLhSjQB9jrzztSw66zMMYDZuxQ",
    "balance": 363.94840658,
    "zcredits": 50
}
```

5.10.1 instantdex_deposit

You can `deposit` KMD using the following script into b addresses which will accrue interest as well. This will allow you to do InstantDEX swaps which finishes a swap within 5-20 seconds. The more KMD you deposit for InstantDEX trading, the more limit you can have. It is a timelocked deposit, keeps the honest traders honest. Gives other traders the assurance that you have a valid deposit to cover any hack attempt.

`instantdex_deposit` the values are 1 to 52 weeks and the amount (minimum of 10.0001). These figures are calibrated to the 5% APR, ie. 1 weeks interest is approx the BOTS 0.1% fee, 52 weeks is the most the 5% APR accrues, 10.0001 KMD is min required for interest. It can take few minutes (about 10 confirmations) for the LP nodes to recognise you. This creates a dynamic trust between nodes and swaps are faster. Just issue normal buy and the LP nodes will respond with InstantDEX swaps.

It creates a binary file in DB with the unique deposit txids on an append only basis. The first time, it seeds it with the contents of the existing `instantdex.json` files. from the deposits file the `instantdex_address_append.json` is directly created and the `instantdex.json` file is created by removing the already spent deposits.

Sample script:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"instantdex_deposit\",\"weeks\":1,\
↪"amount\":10.0001,\"broadcast\":1}"
```

`weeks:1` means it will be there for 300 hours or 1 week time. Don't set to `week:0` unless you are only testing.

Sample Output:

```
{
    "rawtx":
↪"01000000014c9758944771d62c731f9b06246ccd3301ea0e198a6bdbdcb22csdffasdc7010fgd
↪",
    "hex":
↪"01000000014c9758944771d62c731f9b06246ccd3301ea0e1123jdsfbdcb22ce18d5daf0dc701
↪",
    "tx": {
        "version": 1,
        "locktime": 1511458859,
        "vin": [
            {
                "txid":
↪"c70daf5d8de12cb2dcdb6b8a190eea0133cd6c24069b1f732cd671479458974c
↪",
                "vout": 1,
                "scriptPubKey": {
                    "hex":
↪"76a9140c1007fc1f406a0a519886c0e59327e9c43a634088ac"
                }
```

(continues on next page)

(continued from previous page)

```
            }
        ],
        "vout": [
            {
                "satoshis": "1000000000",
                "scriptPubKey": {
                    "hex":
↪"a914d9da8e493573be957273f842617132c361546e7b87"
                }
            },
            {
                "satoshis": "1000003",
                "scriptPubKey": {
                    "hex":
↪"76a914928104f862284623c4e744009923356ff4832ba388ac"
                }
            },
            {
                "satoshis": "124620441",
                "scriptPubKey": {
                    "hex":
↪"76a9140c1007fc1f406a0a519886c0e59327e9c43a634088ac"
                }
            }
        ]
    },
    "txid":
↪"a01b85233259e62edcaf3c171c97de0d2977ce88ae172b9b11809da9757e218c
↪",
    "complete": true,
    "address": "bYbB9P5XG5j1Jr5nbD2nL5jt54WwPEzADd",
    "expiration": 1512201600,
    "deposit": 10,
    "result": "success",
    "broadcast":
↪"a01b85233259e62edcaf3c171c97de0d2977ce88ae172b9b11809da9757e218c"
}
```

5.10.2 claim

For 0conf deposits claim can take extra time. The extra time can be up to a week as it operates on a weekly cycle. Best is to make the 0conf deposit on Saturdays. Friday is safest anywhere in the world.

After the deposit is **expired**, you can claim at any time, though it is rounded up to the next week boundary. You need to use `./claim` API to claim your 0conf deposit (`zcredits`) back. All of your deposits are secured as long as you have the txid handy. If you use `week:0` you need to save the details of timestamp and the `b` address that the funds were sent in order to use with the `./claim` script to claim it back.

```
Minutes or hours is not long enough. Remember, the point here is to␣
↪resolve any blockchain attacks.
```

`claim` API will display the balance of the 0conf deposit `txid` along with interest and wait time to claim. This will claim the deposit instantly if it already expired and ready to claim.

Sample File Contents:

```
curl --url "http://127.0.0.1:7783" --data "{\"userpass\":\"
↪$userpass\",\"method\":\"instantdex_claim\"}"
```

Output:

```
{
    "result": "success",
    "claimed": 0,
    "txids": [
        {
            "txid":
↪"d5705d948a5a4e0171acec3eb718ca1421ef998b37d0af5c37ac3b440898aef5
↪",
            "deposit": 1000,
            "interest": 10.78139269,
            "waittime": 25702320
        }
    ]
}
```

5.10.3 Manually Enable 0conf deposits to work with the GUI

You need to put the `txid` of the 0conf / instantdex deposit in an `instantdex.json` and/or `instantdex_address.json` file inside `DB` dir. If the file is not preset in the DB folder, create the file manually. The DB dir is located `C:/Users/<username>/AppData/Roaming/BarterDEX/DB` in Windows and `~/.BarterDEX/DB` in Linux. The content inside the JSON file should look like following:

1 txid

```
["d5705d948a5a4e0171acec3eb718ca1421ef998b37d0af5c37ac3b440898aef5"]
```

2 or more txid

```
[

↪"d5705d948a5a4e0171acec3eb718ca1421ef998b37d0af5c37ac3b440898aef5
↪",

↪"bd2087d431bb9af6213e73efc58e3384227edcf4827e1cf83f3c153e512a9e1f"
```

(continues on next page)

(continued from previous page)

```
]
```

[Note: If you are using Windows, please use 1 txid at a time. Claim for multiple txid may not work together. Claim single txid each time.]

Then delete any deposit.address binary file for your address i.e.: `deposits.RPZVpjptzfZnFZZoLnuSbfLexjtkhe6uvn`, all this is in DB dir. **Don't delete any dir in here.**

Next time you start BarterDEX, it should then generate a `deposits.<address>` with the binary of the txid and necessary files. If Electrum mode does not work for you, try with native KMD wallet.

```
Negative ``zcredits balance`` is the balance being used for 0conf␣
↪trade, will be available after the next notarized block.
```

CHAPTER 6

Whitepaper

BarterDEX - a truly decentralized exchange

6.1 Abstract

In this whitepaper we describe the implementation of a decentralized exchange that would allow people to trade coins without a counterparty risk. A fully featured service would need to decentralize order matching, trade clearing, and settlement. The order matching would be done through low level pubkey to pubkey messaging protocol and the final settlement through an atomic cross chain protocol. Like any exchange, a decentralized alternative also needs liquidity providers to ensure it has enough liquidity.

6.2 Introduction

Money should be exchanged freely and safely from person to person, and currently the most practical method for it is a central exchange. Such centralized solution requires the funds to be exchanged into IOU tokens that are backed by the exchange service provider. In doing so, the clients are under a constant risk of their assets being stolen either by an inside theft or an outside hack. In order to remove these risks, a decentralized alternative must be established.

Among all the exchanges, the trading tends to be centralized in a few exchanges – and there is a reason for it. The speed advantage of trading the exchange IOU attracts the traders, and the traders attract liquidity, which creates the best prices, which attracts even more traders. This network effect is the reason that a few centralized exchanges have been dominating the trading volumes, while all smaller exchanges, both centralized and decentralized, are suffering from lack of liquidity.

A decentralized trading environment was created in 2014 by a service called MultiGateway, that utilized the Nxt Asset Exchange. With the Nxt Asset Exchange it is possible to do decentralized trading, using proxy tokens to represent external cryptocurrencies like Bitcoin. This hybrid solution is being increasingly used by many other blockchain platforms.

One problem with this type of solution, however, is that it loses the speed of a centralized exchange. And another problem is that the storage of external cryptocurrencies is not decentralized, but only distributed at best, which maintains a degree of counterparty risk for the users. This, combined with the need to use a set of gateways to convert the external, native coins to and from the proxy tokens has made it an impractical solution.

The conclusion is that a decentralized alternative that lacks speed cannot compete with the convenience of existing centralized exchanges, and that simply reducing the counterparty risks it is not enough to attract liquidity.

6.3 The complete solution

BarterDEX combines three key components: order matching, trade clearing and liquidity provision into a single integrated system that allows users to make a coin conversion request, find a suitable match and complete the trade using an atomic cross-chain protocol. Additionally, there is a privacy layer in the order matching so that two nodes can do a peer-to-peer atomic swap without any direct IP contact between them.

Order matching is the process of pairing buy orders with sell orders. It is done by algorithms that define how the orders are paired, and in which order they get filled. A trade is said to become 'executed' once it gets filled.

After execution comes trade clearing, which is the process from the promise of payment to the actual money transfer or settlement, where the assets are swapped between the trading parties.

A liquidity provider (LP) is a trading party that acts as a market maker buying and selling assets. They provide liquidity to the exchange and make their profit from the spread between bid and ask orders. LP's bring price stability and makes it easier for traders to execute trades.

6.4 Order Matching

Before we get into the atomic swap details, there is another aspect of barterDEX that is quite critical: the decentralized orderbook. In order to achieve this, a custom peer to peer network is created that has the analogue of a full relay node and a node that doesn't relay. Network load on this network is minimized using a combination of hierarchical transmission of the orderbooks, along with fetching of data. Also, there are several different methods to obtain data to maximize the number of nodes that are able to fully connect to the barterDEX network.

One aside to mention is that it is possible to create a totally separate set of BarterDEX nodes by seeding the network with a totally independent set of seed nodes. This enables scaling to arbitrary levels as, if any scaling limit is reached, it is a matter to create a separate network that doesn't directly interconnect with the other. At such scale, the assumption is that there is plenty of inventory in the orderbooks, especially if the partitioning is done on a per-reference coin basis. In the event it is desired to cross-pollinate orderbook entries from disparate barterDEX networks, it would be possible to have special bridge nodes that cherry pick the desired entries from one network to the other.

Anybody that wants to can run a full relaying node, there is no specific payment to do this and it does require more bandwidth. However, by being a full relaying node, you have better connectivity with all the other nodes and thus a higher percentage chance of having a trade started and completed. This increase in reliability would be enough for active traders and also for significant owners of the DEX asset, making sure there are enough full nodes is a good thing to do.

A non-relaying node is able to do everything a full relaying node can do, so we expect that the vast majority of nodes will be non-relaying nodes and this will enable the barterDEX network to scale to a large number of total nodes. With 100 full nodes, thousands of non-relay nodes can be supported, possibly tens of thousands, though that number has not been reached in practice, so we will have to wait and see what the real world limitations are.

6.5 Atomic Swaps

BarterDEX implements the Tier Nolan protocol as described in the bitcointalk thread.

6.5.1 Overview of the atomic swap protocol

While the thread is quite technical, it gives a very good background into the tradeoffs that went into selecting the atomic swap protocol. The important thing to note is that at each step of the protocol there are incentives/disincentives to proceed to the next step and that regardless of where the protocol stops, each party ends up with what they should get. The understanding is that if you don’t follow the protocol, you will end up p46 Td (vt)-2i 0 -ethifpa6er.

5. alice spends the bobpayment
6. bob spends the alicepayment
7. bob refunds his own deposit

While it seems a bit inefficient to have 7 transactions for a swap that could be done with just 2 transactions, this is what is required to make it trustless and have the characteristic that at any step, there are incentives to go to the next step and where both sides end up with the right amounts regardless of where things stop.

Let us see what happens if things just stop at a certain step:

1. alice sends dexfee. If bob does not send the bobdeposit, alice is out a dexfee, which is 1/777 of the transaction amount. This will then give bob a bad reputation and very quickly nobody will trade with bob. As long as the frequency of bob failing to deposit is low, the occasional extra dexfee is a minor issue. Contingency plans are in place to provide refunds if a particular alice node experiences a materially large amount of lost dexfees.
2. bob sends bobdeposit. If alice doesn't send the alicepayment, then alice loses not only the dexfee but gains a bad reputation and soon nobody would trade with alice. We don't expect this to happen that often.
3. alice sends alicepayment. If bob doesn't send in payment, after 4 hours, alice can claim the bobdeposit, which is 12.5% larger than the payment, so alice ends up with a nice bonus in this case. I would not be surprised if the alice nodes are eager for this case of atomic swap protocol.
4. bob sends bobpayment. If alice doesn't spend the bobpayment, then after 2 hours he can reclaim his payment and then after 4 hours refund his deposit. Once bob refunds his own deposit, then alice is able to reclaim her payment. It is all intricately interconnected as the spending of a specific transaction enables the other party to spend their counterpart.
5. alice spends the bobpayment. If bob doesn't spend the alicepayment, alice is already done with the trading, so there is no more she needs. Bob is sleeping and doesn't spend the alicepayment, then he is out the alicepayment until he spends it. this is up to bob, but it is a bit dangerous as if he does the refund of the deposit before spending alicepayment, alice would get the info needed to reclaim her payment. It is important for both participants to continue running the atomic swap protocol until it completes. If after 4 hours, bob is still sleeping, then alice is able to claim the deposit and become a happy camper. alice spending the bob deposit, however, does not give her the information needed to reclaim her own payment, so bob is still able to do this when he wakes up.
6. bob spends the alicepayment. Similar to above, if bob doesn't refund his own deposit, it is his loss and purely his responsibility. If after 4 hours, he still hasnt, then alice will be able to claim the deposit.
7. bob refunds his own deposit. As you can see at each step, the side that needs to do something is motivated to do so, with greater and greater urgency toward the end.

BarterDEX implements the above in a cross-platform way, across almost a hundred coins, using either native coin daemon or SPV electrum servers. A swap that is not completed during one session can be completed as long as BarterDEX is run before the time expires. It is best to not trade a very large amount unless you are sure of your node's reliability, especially regarding the internet connection.

Believe it or not, doing the above atomic swap protocol with all the cryptographic validations in between along with a fancy key exchange, is less than half the difficulty of barterDEX. Relatively speaking, it is 'easy' to do an atomic swap in isolation between two test nodes with carefully prepared UTXOs made especially for the test. It is an entirely different matter to be able to let anybody start trading with anybody else and have things like orderbooks and ordermatching happen. Due to the peer to peer nature, it is impossible to guarantee a successful swap, however, a failed swap to start is just a few seconds of lost time and there is no cost to try to start a swap. Just like in normal trading, there is no guarantee that you can get the trade you want, similarly, there is no guarantee with barterDEX.

6.5.2 Details of the atomic swap protocol

Let us see what is required in a bit more detail as we now have the context of the atomic swap protocol. In order to even start an atomic swap, there needs to be a pair from alice such that the dexfee and alicepayment can be created and also two from bob, such that the bobdeposit and bobpayment can be created. BarterDEX requires all four of these UTXOs to be specified before the start of the atomic swap protocol.

Here comes the first user issue. Most users don't know what an UTXO is and most view their balance as a single blob of coins they can spend at the satoshi level. The reality is that the bitcoin protocol maintains a list of Unspent Transaction Outputs (UTXO) of specific values and to make a transaction, there needs to be inputs sufficient to pay for the outputs. Any excess goes into a change output (let us ignore txfees for now, even though BarterDEX automatically calculates the current BTC txfee to pay that will get confirmed quickly).

It is impractical to have the user specify which UTXO pair to use, and it is not possible for alice to even know what UTXOs bob has available at the moment of negotiating a trade. What barterDEX does is an atomic swap negotiation protocol as follows:

1. alice sends a "request" to a specific bob with her pair of UTXOs, price and volume
2. bob validates the "request" to make sure the alice UTXOs are valid and that the price is acceptable, then bob scans all his UTXOs for the most efficient way to create both the bobpayment and bobdeposit UTXOs. The constraint is that it needs to match the price alice wants to pay and the volume and the deposit at least 12.5% bigger than the payment. If all these things are met, then bob sends back a "reserved" packet to alice. By doing this just in time, bob minimizes the funds that are tied up doing deposit duty.
3. alice validates the "reserved" packet from bob, making sure all the UTXOs are valid, the price and volumes are acceptable and if so sends a "connect" packet to bob with the same parameters as in the "reserved". Between the "request" being sent and the "reserved" being received or a 10-seconds timeout, alice is prevented from making any other trade request. It is important to make sure the current pending atomic swap is properly started and this prevention is also part of the whale resistance property for dICO (decentralized initial coin offering).
4. bob validates the "connect" and if all is well, starts a new thread to do the atomic swap.

5. alice receives the "connect" and if all is well, starts a new thread to do the atomic swap.

There is one more "negotiation" step that is needed between alice and bob and while it could have been part of the 5 steps above, due to legacy reasons it ended up inside the atomic swap protocol itself. In the event there is no consensus on the coin confirmations to use, the atomic swap aborts without any payments being sent. No harm, no foul.

6.5.3 DEX fee - dexfee

People will notice that there is a small dexfee, paid by alice, as part of the barterDEX protocol. This is 1/777 of the transaction amount and it is calibrated to make spam attacks impractical. Impractical as in costing real money. Without this spam prevention, BarterDEX could be cost-effectively DOS attacked at the protocol level.

The 1/777 ends up being 0.1287%. This is less than almost all the centralized exchanges, in many cases by a significant margin. Please note that the central exchanges charge both sides of the trade, so even if they charge 0.2%, it is actually 0.4% total fees.

The dexfee helps secure the barterDEX network and it is set at a level that is less than the central exchanges. It is possible that some trades can start without completing and since the dexfee is charged first in the protocol. In this sense, there would be a dexfee charged for these failed atomic swaps. While the decentralized exchanges using proxies charge for every bid and every ask, even if a trade doesn't happen at all. This is a big negative for trading on proxy-powered DEX's.

However, it should not be looked upon in isolation. The barterDEX protocol is based on statistics and statistically, there will be some percentage of atomic swaps that are started that won't complete. Let us say this is a 15% failure rate (this is much higher than we are seeing in testing, where 95%+ of atomic swaps that are started complete, or reach the bobdeposit phase at the minimum), then the effective dexfee cost is still around 0.15%

This is why even though the 1/777 ratio is 0.1287%, it is better to state the dexfee as 0.15%

Fees are collected and distributed to DEX asset holders, which is an assetchain. Any DEX assetchain unit holder will receive this DEX fee as a dividend, paid in KMD.

6.5.4 Transaction Confirmations

Since barterDEX is trading real coins and not just updating an internal database (or a proxy tokens account balance for proxy DEX), both sides need to wait for coin confirmations to the level they are comfortable. Since the payments sent on one chain won't be reorganized if the other chain does, it is important to have enough confirmations for the size of the trade being done. In order to enable this, there is a setconfirms API call that can be called for each coin. This needs to be done before the atomic swap is started as the current numconfirms for the coin will be sent to the other side and the larger of alice or bob's numconfirms will be used. There is also a maxconfirms value to prevent one side from specifying an abnormal high amount of numconfirms.

6.5.5 Zero Transaction Confirmations

To be able to offer the user a trading experience similar to existing centralized exchanges, zero confirmation mode can be enabled. It is quite risky to do this, especially for fast block confirmation time coins with low hashrate. For zero confirmation swaps to be enabled, both sides of a trade need to have so-called zcredits, a locked amount of KMD that is more than the value of that trade.

6.6 Efficiency of orderbook

We have now worked backward from the atomic swap details to the ordermatch process and this leaves the efficient orderbook propagation as the only part left to describe. barterDEX uses a convention of base/rel meaning base currency converted to rel currency. Buying a base/rel means to use rel currency to buy base currency, price denominated in base/rel: buy 1 base for x rel, x being the price.

In order to construct an orderbook a node needs to have price information and since everything is pubkey based, this means a price from a pubkey. Ultimately a specific txid/vout (UTXO) is needed, but a single node could have hundreds of UTXOs and this would use up a lot of network bandwidth to propagate it globally. barterDEX, therefore, uses a hierarchical orderbook, where the skeleton of it is just the pubkey/price for a particular base/rel pair. Note that a buy of base/rel at price is the same as a sell of rel/base at 1/price. So all that is needed to populate the orderbook skeleton is for a node to broadcast its pubkey and price for a base/rel pair. Given this, nodes that are running a local coin daemon can find the possible list of UTXOs via listunspent on demand.

Critical information is fully signed to prevent spoofing, so all nodes can verify the smartaddress associated with a pubkey and also that the price being broadcast is a valid price. The electrum SPV coins do a specific SPV validation for all UTXO before they are approved for trading.

If all nodes were always broadcasting all their UTXOs to all other nodes, it would rapidly lead to congestion. Most of the time barterDEX just relies on the pubkey/prices and this is enough to create useful orderbooks. Since there are N*N possible orderbooks given N currencies, it is not practical to be updating all possible orderbooks, instead, they are created when requested from the raw data. During the orderbook creation, if the top entries in the orderbook don't have any listunspent data, a request for it is made to the network.

This process ensures that by the time a trade is done, already an orderbook has been requested which in turn requests the listunspent data for the most likely pubkeys. The actual ordermatch process then iterates through the orderbook scanning all the locally known UTXOs to find a high probability counterparty to make the "request" offer to. In practice, we are seeing the nearly instantaneous response when all the parameters are properly met.

6.7 References

barterDEX – A Practical Native DEX (https://github.com/KomodoPlatform/KomodoPlatform/wiki/BarterDEX-%E2%80%93-A-Practical-Native-DEX)

Nakamoto Satoshi (2008): Bitcoin: A peer-to-peer electronic cash system. (http://www.bitcoin.org/bitcoin.pdf)

Mtchl (2014): The math of Nxt forging (https://www.docdroid.net/ahms/forging0-4-1.pdf.html)

King Sunny, Nadal Scott (2012): PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake (https://peercoin.net/assets/paper/peercoin-paper.pdf)

Delegated Proof-of-Stake Consensus (https://bitshares.org/technology/delegated-proof-of-stake-consensus/)

Miers Ian, Garman Christina, Green Matthew, Rubin Aviel: Zerocoin: Anonymous Distributed E-Cash from Bitcoin (https://isi.jhu.edu/~mgreen/ZerocoinOakland.pdf)

Ben-Sasson Eli, Chiesa Alessandro, Garman Christina, Green Matthew, Miers Ian, Troer Eran, Virza Madars (2014): Zerocash: Decentralized Anonymous Payments from Bitcoin (http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf)

Ben-Sasson Eli, Chiesa Alessandro, Green Matthew, Tromer Eran, Virza Madars (2015): Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs (https://www.diyhpl.us/~bryan/papers2/bitcoin/snarks/Secure%20sampling%20of%20public%20parameters%20for%20succinct%20zero%20knowledge%20proofs.pdf)

NXT Community: NXT Whitepaper (https://nxtwiki.org/wiki/Whitepaper:Nxt)

Larimer Daniel, Scott Ned, Zavgorodnev Valentine, Johnson Benjamin, Calfee James, Vandeberg

Michael (March 2016): Steem, An incentivized, blockchain-based social media platform.(https://steem.io/SteemWhitePaper.pdf)

BitFury Group (Sep 13, 2015): Proof of Stake versus Proof of Work White Paper (http://bitfury.com/content/5-white-papers-research/pos-vs-pow-1.0.2.pdf)