Options available to the Verus RPC client.

Important General Information

Verus CLI version v1.2.0-1

Usage:

verus [options] [command] Issue a command to the coindaemon
verus [options] help List commands
verus [options] help <command> Get help for a command

Verus Wallet.dat, Chaindata & VRSC.conf standard locations

Linux: ~/.komodo/VRSC
Mac OS: ~/Library/Application Support/Komodo/VRSC
Windows 10: %AppData%\Roaming\Komodo\VRSC\

Options:

-? This help message

-conf=<file> Specify configuration file (default: komodo.conf)

-datadir=<dir> Specify data directory

-testnet Use the test network

-regtest Enter regression test mode, which uses a special chain in which blocks can be solved instantly. This is intended for regression testing tools and app development.

-rpcconnect=<ip> Send commands to node running on (default: 127.0.0.1)

-rpcport=<port> Connect to JSON-RPC on (default: 8232 or testnet: 18232)

-rpcwait Wait for RPC server to start

-rpcuser=<user> Username for JSON-RPC connections

-rpcpassword=<pw> Password for JSON-RPC connections

-rpcclienttimeout=<n> Timeout in seconds during HTTP requests, or 0 for no timeout. (default: 900)

-stdin Read extra arguments from standard input, one per line until EOF/Ctrl-D (recommended for sensitive information such as passphrases)

Commands:

Addressindex

getaddressbalance

Returns the balance for an address(es) (requires addressindex to be enabled).

Arguments:

```
{
  "addresses"
    [
      "address"  (string) The base58check encoded address
      ,...
    ]
}
```

Result:

```
{
  "balance"  (string) The current balance in satoshis
  "received"  (string) The total number of satoshis received (including change)
}
```

Examples:

```
> verus getaddressbalance '{"addresses": ["RY5LccmGiX9bUHYGtSWQouNy1yFhc5rM87"]}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getaddressbalance", "param
s": [{"addresses": ["RY5LccmGiX9bUHYGtSWQouNy1yFhc5rM87"]}] }' -H 'content-type: text/plain;' http://127.0.0.1:27
486/
```

getaddressdeltas

Returns all changes for an address (requires addressindex to be enabled).

Arguments:

```
{
  "addresses"
    [
      "address"  (string) The base58check encoded address
      ,...
    ]
  "start" (number) The start block height
  "end" (number) The end block height
  "chainInfo" (boolean) Include chain info in results, only applies if start and end specified
  "friendlynames" (boolean) Include additional array of friendly names keyed by currency i-addresses
  "verbosity" (number) (default == 0), if 1, include output information for spends, including all reserve amounts
and destinations
}
```

Result:

```
[
  {
    "satoshis"  (number) The difference of satoshis
    "txid"  (string) The related txid
    "index"  (number) The related input or output index
    "height"  (number) The block height
    "address"  (string) The base58check encoded address
  }
]
```

Examples:

```
> verus getaddressdeltas '{"addresses": ["RY5LccmGiX9bUHYGtSWQouNy1yFhc5rM87"]}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getaddressdeltas", "param
s": [{"addresses": ["RY5LccmGiX9bUHYGtSWQouNy1yFhc5rM87"]}] }' -H 'content-type: text/plain;' http://127.0.0.1:27
486/
```

getaddressmempool

Returns all mempool deltas for an address (requires addressindex to be enabled).

Arguments:

```
{
  "addresses"
    [
      "address"  (string) The base58check encoded address
      ,...
    ]
    "friendlynames"    (boolean) Include additional array of friendly names keyed by currency i-addresses
    "verbosity"        (number) (default == 0), if 1, include output information for spends, including all reserv
e amounts and destinations
}
```

Result:

```
[
  {
    "address"  (string) The base58check encoded address
    "txid"  (string) The related txid
    "index"  (number) The related input or output index
    "satoshis"  (number) The difference of satoshis
    "timestamp"  (number) The time the transaction entered the mempool (seconds)
    "prevtxid"  (string) The previous txid (if spending)
    "prevout"  (string) The previous transaction output index (if spending)
  }
]
```

Examples:

```
> verus getaddressmempool '{"addresses": ["RY5LccmGiX9bUHYGtSWQouNy1yFhc5rM87"]}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getaddressmempool", "param
s": [{"addresses": ["RY5LccmGiX9bUHYGtSWQouNy1yFhc5rM87"]}] }' -H 'content-type: text/plain;' http://127.0.0.1:27
486/
```

getaddresstxids

Returns the txids for an address(es) (requires addressindex to be enabled).

Arguments:

```
{
  "addresses"
    [
      "address"  (string) The base58check encoded address
      ,...
    ]
  "start" (number) The start block height
  "end" (number) The end block height
}
```

Result:

```
[
  "transactionid"  (string) The transaction id
  ,...
]
```

Examples:

```
> verus getaddresstxids '{"addresses": ["RY5LccmGiX9bUHYGtSWQouNy1yFhc5rM87"]}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getaddresstxids", "param
s": [{"addresses": ["RY5LccmGiX9bUHYGtSWQouNy1yFhc5rM87"]}] }' -H 'content-type: text/plain;' http://127.0.0.1:27
486/
```

getaddressutxos

Returns all unspent outputs for an address (requires addressindex to be enabled).

Arguments:

```
{
  "addresses"
    [
      "address"  (string) The base58check encoded address
      ,...
    ],
  "chainInfo"  (boolean) Include chain info with results
  "friendlynames" (boolean) Include additional array of friendly names keyed by currency i-addresses
  "verbosity"    (number) (default == 0), if 1, include output information for spends, including all reserve amou
nts and destinations
}
```

Result:

```
[
  {
    "address"  (string) The address base58check encoded
    "txid"  (string) The output txid
    "height"  (number) The block height
    "outputIndex"  (number) The output index
    "script"  (strin) The script hex encoded
    "satoshis"  (number) The number of satoshis of the output
  }
]
```

Examples:

```
> verus getaddressutxos '{"addresses": ["RY5LccmGiX9bUHYGtSWQouNy1yFhc5rM87"]}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getaddressutxos", "param
s": [{"addresses": ["RY5LccmGiX9bUHYGtSWQouNy1yFhc5rM87"]}] }' -H 'content-type: text/plain;' http://127.0.0.1:27
486/
```

getsnapshot

Returns a snapshot of (address,amount) pairs at current height (requires addressindex to be enabled).

Arguments:

"top" (number, optional) Only return this many addresses, i.e. top N richlist

Result:

```
{
   "addresses": [
    {
      "addr": "RMEBhzvATA8mrfVK82E5TgPzzjtaggRGN3",
      "amount": "100.0"
    },
    {
      "addr": "RqEBhzvATAJmrfVL82E57gPzzjtaggR777",
      "amount": "23.45"
    }
  ],
  "total": 123.45           (numeric) Total amount in snapshot
  "average": 61.7,          (numeric) Average amount in each address
  "utxos": 14,              (number) Total number of UTXOs in snapshot
  "total_addresses": 2,     (number) Total number of addresses in snapshot,
  "start_height": 91,       (number) Block height snapshot began
  "ending_height": 91       (number) Block height snapsho finished,
  "start_time": 1531982752, (number) Unix epoch time snapshot started
  "end_time": 1531982752    (number) Unix epoch time snapshot finished
}
```

Examples:

```
> verus getsnapshot
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getsnapshot", "params": [1
000] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

Blockchain

coinsupply <height>

Return coin supply information at a given block height. If no height is given, the current height is used.

Arguments:

1. "height" (integer, optional) Block height

Result:

```
{
  "result" : "success",         (string) If the request was successful.
  "coin" : "VRSC",              (string) The currency symbol of the native coin of this blockchain.
  "height" : 420,                 (integer) The height of this coin supply data
  "supply" : "777.0",           (float) The transparent coin supply
  "zfunds" : "0.777",           (float) The shielded coin supply (in zaddrs)
  "total" :  "777.777",         (float) The total coin supply, i.e. sum of supply + zfunds
}
```

Examples:

```
> verus coinsupply 420
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "coinsupply", "params": [42
0] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

getbestblockhash

Returns the hash of the best (tip) block in the longest block chain.

Result:

```
"hex"      (string) the block hash hex encoded
```

Examples

```
> verus getbestblockhash
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getbestblockhash", "param
s": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

getblock "hash|height" (verbosity)

If verbosity is 0, returns a string that is serialized, hex-encoded data for the block.
If verbosity is 1, returns an Object with information about the block.
If verbosity is 2, returns an Object with information about the block and information about each transaction.

Arguments:

1. "hash|height" (string, required) The block hash or height
2. verbosity (numeric, optional, default=1) 0 for hex encoded data, 1 for a json object, and 2 for json object with transaction data

Result (for verbosity = 0):
"data" (string) A string that is serialized, hex-encoded data for the block.

Results (for verbosity = 1):

```
{
  "hash" : "hash",       (string) the block hash (same as provided hash)
  "confirmations" : n,   (numeric) The number of confirmations, or -1 if the block is not on the main chain
  "size" : n,            (numeric) The block size
  "height" : n,          (numeric) The block height or index (same as provided height)
  "version" : n,         (numeric) The block version
  "merkleroot" : "xxxx", (string) The merkle root
  "finalsaplingroot" : "xxxx", (string) The root of the Sapling commitment tree after applying this block
  "tx" : [               (array of string) The transaction ids
     "transactionid"     (string) The transaction id
     ,...
  ],
  "time" : ttt,          (numeric) The block time in seconds since epoch (Jan 1 1970 GMT)
  "nonce" : n,           (numeric) The nonce
  "bits" : "1d00ffff",   (string) The bits
  "difficulty" : x.xxx,  (numeric) The difficulty
  "previousblockhash" : "hash",  (string) The hash of the previous block
  "nextblockhash" : "hash"       (string) The hash of the next block
}
```

Result (for verbosity = 2):

```
{
  ...,                     Same output as verbosity = 1.
  "tx" : [               (array of Objects) The transactions in the format of the getrawtransaction RPC. Differen
t from verbosity = 1 "tx" result.
         ,...
  ],
  ,...                     Same output as verbosity = 1.
}
```

Examples:

```
> verus getblock "00000000febc373a1da2bd9f887b105ad79ddc26ac26c2b28652d64e5207c5b5"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getblock", "params": ["00000000febc373a1da2bd9f887b105ad79ddc26ac26c2b28652d64e5207c5b5"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
> verus getblock 12800
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getblock", "params": [12800] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getblockchaininfo`

Returns an object containing various state info regarding block chain processing.

Note that when the chain tip is at the last block before a network upgrade activation,
consensus.chaintip != consensus.nextblock.

Result:

```
{
  "chain": "xxxx",              (string) current network name as defined in BIP70 (main, test, regtest)
  "name": "xxxx",               (string) current network name of blockchain ID (VRSC, VRSCTEST, PBAASNAME)
  "chainid": "xxxx",            (string) blockchain ID (i-address of the native blockchain currency)
  "blocks": xxxxxx,             (numeric) the current number of blocks processed in the server
  "headers": xxxxxx,            (numeric) the current number of headers we have validated
  "bestblockhash": "...",       (string) the hash of the currently best block
  "difficulty": xxxxxx,         (numeric) the current difficulty
  "verificationprogress": xxxx, (numeric) estimate of verification progress [0..1]
  "chainwork": "xxxx"           (string) total amount of work in active chain, in hexadecimal
  "chainstake": "xxxx"          (string) total amount of stake in active chain, in hexadecimal
  "pruned": "xx"                (boolean) when true the local chain is pruned
  "size_on_disk": xxxxxx,       (numeric) the estimated size of the block and undo files on disk
  "commitments": xxxxxx,        (numeric) the current number of note commitments in the commitment tree
  "valuePools": [               (array) status of valuepools
    {
      "id": "xxxx",             (string) name of valuepool
      "monitored": xx,          (boolean)
      "chainValue": xxxx,       (numeric) total coins in this valuepool
      "chainValueZat": xxxx,    (numeric) total sats in this valuepool
    },
  ],
  "softforks": [                 (array) status of softforks in progress
     {
        "id": "xxxx",            (string) name of softfork
        "version": xx,           (numeric) block version
        "enforce": {             (object) progress toward enforcing the softfork rules for new-version blocks
           "status": xx,         (boolean) true if threshold reached
           "found": xx,          (numeric) number of blocks with the new version found
           "required": xx,       (numeric) number of blocks required to trigger
           "window": xx,         (numeric) maximum size of examined window of recent blocks
        },
        "reject": { ... }        (object) progress toward rejecting pre-softfork blocks (same fields as "enforce")
     }, ...
  ],
  "upgrades": {                  (object) status of network upgrades
     "xxxx" : {                  (string) branch ID of the upgrade
        "name": "xxxx",          (string) name of upgrade
        "activationheight": xxxxxx,  (numeric) block height of activation
        "status": "xxxx",        (string) status of upgrade
        "info": "xxxx",          (string) additional information about upgrade
     }, ...
  },
  "consensus": {                 (object) branch IDs of the current and upcoming consensus rules
     "chaintip": "xxxxxxxx",     (string) branch ID used to validate the current chain tip
     "nextblock": "xxxxxxxx"     (string) branch ID that the next block will be validated under
  }
}
```

`getblockcount`

Returns the number of blocks in the best valid block chain.

Result:

```
n    (numeric) The current block count
```

Examples:

```
> verus getblockcount
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getblockcount", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getblockdeltas '["blockhash"]'`

Returns information about the given block and its transactions.

WARNING: getblockdeltas is disabled.
To enable it, restart verusd with the `-experimentalfeatures` and `-insightexplorer` commandline options, or add these two lines to the `VRSC.conf` file:

```
experimentalfeatures=1
insightexplorer=1
```

Arguments:

1. `["hash"]` (json, required) The block hash

Result:

```
{
  "hash": "hash",              (string) block ID
  "confirmations": n,          (numeric) number of confirmations
  "size": n,                   (numeric) block size in bytes
  "height": n,                 (numeric) block height
  "version": n,                (numeric) block version (e.g. 4)
  "merkleroot": "hash",        (hexstring) block Merkle root
  "deltas": [
    {
      "txid": "hash",          (hexstring) transaction ID
      "index": n,              (numeric) The offset of the tx in the block
      "inputs": [                (array of json objects)
        {
          "address": "taddr",  (string) transparent address
          "satoshis": n,       (numeric) negative of spend amount
          "index": n,          (numeric) vin index
          "prevtxid": "hash",  (string) source utxo tx ID
          "prevout": n         (numeric) source utxo index
        }, ...
      ],
      "outputs": [             (array of json objects)
        {
          "address": "taddr",  (string) transparent address
          "satoshis": n,       (numeric) amount
          "index": n           (numeric) vout index
        }, ...
      ]
    }, ...
  ],
  "time" : n,                  (numeric) The block version
  "mediantime": n,             (numeric) The most recent blocks' ave time
  "nonce" : "nonce",           (hex string) The nonce
  "bits" : "1d00ffff",         (hex string) The bits
  "difficulty": n,             (numeric) the current difficulty
  "chainwork": "xxxx"          (hex string) total amount of work in active chain
  "previousblockhash" : "hash",(hex string) The hash of the previous block
  "nextblockhash" : "hash"     (hex string) The hash of the next block
}
```

Examples:

```
> verus getblockdeltas 00227e566682aebd6a7a5b772c96d7a999cadaebeaf1ce96f4191a3aad58b00b
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getblockdeltas", "params": ["00227e566682aebd6a7a5b772c96d7a999cadaebeaf1ce96f4191a3aad58b00b"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getblockhash index`

Returns hash of block in best-block-chain at index provided.

Arguments:

```
index        (numeric, required) The block index
```

Result:

```
"hash"         (string) The block hash
```

Examples:

```
> verus getblockhash 1000
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getblockhash", "params": [1000] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getblockhashes timestamp`

Returns array of hashes of blocks within the timestamp range provided.

Arguments:

```
1. high         (numeric, required) The newer block timestamp
2. low          (numeric, required) The older block timestamp
3. options      (string, required) A json object
    {
      "noOrphans":true   (boolean) will only include blocks on the main chain
      "logicalTimes":true   (boolean) will include logical timestamps with hashes
    }
```

Result:

```
[
  "hash"         (string) The block hash
]
[
  {
    "blockhash": (string) The block hash
    "logicalts": (numeric) The logical timestamp
  }
]
```

Examples:

```
> verus getblockhashes 1231614698 1231024505
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getblockhashes", "params": [1231614698, 1231024505] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
> verus getblockhashes 1231614698 1231024505 '{"noOrphans":false, "logicalTimes":true}'
```

`getblockheader "hash" ( verbose )`

If verbose is false, returns a string that is serialized, hex-encoded data for blockheader 'hash'.
If verbose is true, returns an Object with information about blockheader .

Arguments:

```
1. "hash"          (string, required) The block hash
2. verbose           (boolean, optional, default=true) true for a json object, false for the hex encoded data
```

Result (for verbose = true):

```
{
  "hash" : "hash",       (string) the block hash (same as provided)
  "confirmations" : n,   (numeric) The number of confirmations, or -1 if the block is not on the main chain
  "height" : n,          (numeric) The block height or index
  "version" : n,         (numeric) The block version
  "merkleroot" : "xxxx", (string) The merkle root
  "finalsaplingroot" : "xxxx", (string) The root of the Sapling commitment tree after applying this block
  "time" : ttt,          (numeric) The block time in seconds since epoch (Jan 1 1970 GMT)
  "nonce" : n,           (numeric) The nonce
  "bits" : "1d00ffff",   (string) The bits
  "difficulty" : x.xxx,  (numeric) The difficulty
  "previousblockhash" : "hash",  (string) The hash of the previous block
  "nextblockhash" : "hash"       (string) The hash of the next block
}
```

Result (for verbose=false):

```
"data"             (string) A string that is serialized, hex-encoded data for block 'hash'.
```

Examples:

```
> verus getblockheader "00000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getblockheader", "params": ["00000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getchaintips`

Return information about all known tips in the block tree, including the main chain as well as orphaned branches.

Result:

```
[
  {
    "height": xxxx,         (numeric) height of the chain tip
    "hash": "xxxx",         (string) block hash of the tip
    "branchlen": 0          (numeric) zero for main chain
    "status": "active"      (string) "active" for the main chain
  },
  {
    "height": xxxx,
    "hash": "xxxx",
    "branchlen": 1          (numeric) length of branch connecting the tip to the main chain
    "status": "xxxx"        (string) status of the chain (active, valid-fork, valid-headers, headers-only, invalid)
  }
]
```

Possible values for status:
1. "invalid" This branch contains at least one invalid block
2. "headers-only" Not all blocks for this branch are available, but the headers are valid
3. "valid-headers" All blocks are available for this branch, but they were never fully validated
4. "valid-fork" This branch is not part of the active chain, but is fully validated
5. "active" This is the tip of the active main chain, which is certainly valid

Examples:

```
> verus getchaintips
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getchaintips", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getchaintxstats`

Compute statistics about the total number and rate of transactions in the chain.

Arguments:

1. nblocks (numeric, optional) Number of blocks in averaging window.
2. blockhash (string, optional) The hash of the block which ends the window.

Result:

```
{
  "time": xxxxx,                         (numeric) The timestamp for the final block in the window in UNIX format.
  "txcount": xxxxx,                      (numeric) The total number of transactions in the chain up to that point.
  "window_final_block_hash": "...",      (string) The hash of the final block in the window.
  "window_block_count": xxxxx,           (numeric) Size of the window in number of blocks.
  "window_tx_count": xxxxx,              (numeric) The number of transactions in the window. Only returned if "window_block_count" is > 0.
  "window_interval": xxxxx,              (numeric) The elapsed time in the window in seconds. Only returned if "window_block_count" is > 0.
  "txrate": x.xx,                        (numeric) The average rate of transactions per second in the window. Only returned if "window_interval" is > 0.
}
```

Examples:

```
> verus getchaintxstats
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getchaintxstats", "params": [2016] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getdifficulty`

Returns the proof-of-work difficulty as a multiple of the minimum difficulty.

Result:

```
n.nnn       (numeric) the proof-of-work difficulty as a multiple of the minimum difficulty.
```

Examples:

```
> verus getdifficulty
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getdifficulty", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getmempoolinfo`

Returns details on the active state of the TX memory pool.

Result:

```
{
  "size": xxxxx               (numeric) Current tx count
  "bytes": xxxxx              (numeric) Sum of all tx sizes
  "usage": xxxxx              (numeric) Total memory usage for the mempool
}
```

Examples:

```
> verus getmempoolinfo
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getmempoolinfo", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getrawmempool ( verbose )`

Returns all transaction ids in memory pool as a json array of string transaction ids.

Arguments:

1. verbose (boolean, optional, default=false) true for a json object, false for array of transaction ids

Result (for verbose = false):

```
[                     (json array of string)
  "transactionid"     (string) The transaction id
  ,...
]
```

Result (for verbose = true):

```
{                           (json object)
  "transactionid" : {       (json object)
```

```
  "size" : n,                  (numeric) transaction size in bytes
  "fee" : n,                   (numeric) transaction fee in satoshis
  "time" : n,                  (numeric) local time transaction entered pool in seconds since 1 Jan 1970 GMT
  "height" : n,                (numeric) block height when transaction entered pool
  "startingpriority" : n,      (numeric) priority when transaction entered pool
  "currentpriority" : n,       (numeric) transaction priority now
  "depends" : [                (array) unconfirmed transactions used as inputs for this transaction
     "transactionid",          (string) parent transaction id
     ...
  ], ...
}
```

Examples:

```
> verus getrawmempool true
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getrawmempool", "params": [true] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

getspentinfo

Returns the txid and index where an output is spent.

Arguments:

```
{
  "txid" (string) The hex string of the txid
  "index" (number) The start block height
}
```

Result:

```
{
  "txid" (string) The transaction id
  "index" (number) The spending input index
  ,...
}
```

Examples:

```
> verus getspentinfo '{"txid": "0d37cd7f6f21caaa823d2f59ca8ba3388b5f2d85fa9c36fd82d38aa5a97f5fd", "index": 0}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getspentinfo", "params": [{"txid": "0d37cd7f6f21caaa823d2f59ca8ba3388b5f2d85fa9c36fd82d38aa5a97f5fd", "index": 0}] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

gettxout "txid" n (includemempool)

Returns details about an unspent transaction output.

Arguments:

```
1. "txid"           (string, required) The transaction id
2. n                (numeric, required) vout value
3. includemempool   (boolean, optional) Whether to include the mempool
```

Result:

```
{
  "bestblock" : "hash",    (string) the block hash
  "confirmations" : n,     (numeric) The number of confirmations
  "value" : x.xxx,         (numeric) The transaction value in VRSC
  "scriptPubKey" : {       (json object)
    "asm" : "code",        (string)
    "hex" : "hex",         (string)
    "reqSigs" : n,         (numeric) Number of required signatures
    "type" : "pubkeyhash", (string) The type, eg pubkeyhash
    "addresses" : [        (array of string) array of Komodo addresses
      "verusaddress"       (string) Komodo address
      ,...
    ]
  },
  "version" : n,           (numeric) The version
  "coinbase" : true|false  (boolean) Coinbase or not
}
```

Examples:

Get unspent transactions

```
> verus listunspent
```

View the details

```
> verus gettxout "txid" 1
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "gettxout", "params": ["txid", n] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

gettxoutproof ["txid",...] (blockhash)

Returns a hex-encoded proof that "txid" was included in a block.

NOTE: By default this function only works sometimes. This is when there is an unspent output in the utxo for this transaction. To make it always work, you need to maintain a transaction index, using the -txindex command line option or specify the block in which the transaction is included in manually (by blockhash).

Return the raw transaction data.

Arguments:

```
1. "txids"       (string) A json array of txids to filter
   [
     "txid"     (string) A transaction hash
     ,...
   ]
```

2. "block hash" (string, optional) If specified, looks for txid in the block with this hash

Result:

```
"data"           (string) A string that is a serialized, hex-encoded data for the proof.
```

gettxoutsetinfo

Returns statistics about the unspent transaction output set.
Note this call may take some time.

Result:

```
{
  "height":n,                 (numeric) The current block height (index)
  "bestblock": "hex",         (string) the best block hash hex
  "transactions": n,          (numeric) The number of transactions
  "txouts": n,                (numeric) The number of output transactions
  "bytes_serialized": n,      (numeric) The serialized size
  "hash_serialized": "hash",  (string) The serialized hash
  "total_amount": x.xxx       (numeric) The total amount
}
```

Examples:

```
> verus gettxoutsetinfo
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "gettxoutsetinfo", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

kvsearch key

Search for a key stored via the kvupdate command. This feature is only available for asset chains.

Arguments:

```
1. key                      (string, required) search the chain for this key
```

Result:

```
{
  "txid": "xxxxx",          (string) chain the key is stored on
  "currentheight": xxxxx,   (numeric) current height of the chain
  "key": "xxxxx",           (string) key
  "keylen": xxxxx,          (string) length of the key
  "owner": "xxxxx",         (string) the address representing the owner of the key
  "height": xxxxx,          (numeric) height the key was stored at
  "expiration": xxxxx,      (numeric) height the key will expire
  "flags": n,               (numeric) 2 if the key was created with a passwords, 0 otherwise.
  "value": "xxxxx",         (string) stored value
  "valuesize": xxxxx        (string) amount of characters stored
}
```

Examples:

```
> verus kvsearch examplekey
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "kvsearch", "params": [examplekey] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

kvupdate key "value" days passphrase

Store a key value. This feature is only available for asset chains.

Arguments:

```
1. key                      (string, required) key
2. "value"                  (string, required) value
3. days                     (numeric, required) amount of days(1440 blocks/day) before the key expires. Minimum 1 day
4. passphrase               (string, optional) passphrase required to update this key
```

Result:

```
{
  "txid": "xxxxx",          (string) chain the key is stored on
  "height": xxxxx,          (numeric) height the key was stored at
  "expiration": xxxxx,      (numeric) height the key will expire
  "flags": n,               (string) amount of days the key will be stored
  "key": "xxxxx",           (numeric) stored key
  "keylen": xxxxx,          (numeric) length of the key
  "value": "xxxxx"          (numeric) stored value
  "valuesize": xxxxx,       (string) amount of characters stored
  "fee": xxxxx              (string) transaction fee paid to store the key
  "txid": "xxxxx"           (string) transaction id
}
```

Examples:

```
> verus kvupdate examplekey "examplevalue" 2 examplepassphrase
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "kvupdate", "params": [examplekey "examplevalue" 2 examplepassphrase] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

minerids needs height

notaries height timestamp

processupgradedata (upgradedata)

Returns the txid and index where an output is spent.

Arguments:

```
{
  "upgradeid"               (string) The VDXF key identifier
  "minimumdemotxversion"    (string) The minimum version required for the upgrade
  "activationheight"        (number) The block height to activate
  "activationtime"          (number) Epoch time to activate, depending on upgrade
}
```

Result:

```
{
  "txid" (string) The transaction id
  "index" (number) The spending input index
  ,...
}
```

Examples:

```
> verus processupgradedata '{"txid": "0d37cd7f6f21caaa823d2f59ca8ba3388b5f2d85fa9c36fd82d38aa5a97f5fd", "index": 0}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "processupgradedata", "params": [{"txid": "0d37cd7f6f21caaa823d2f59ca8ba3388b5f2d85fa9c36fd82d38aa5a97f5fd", "index": 0}] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

verifychain (checklevel numblocks)

Verifies blockchain database.

Arguments:

```
1. checklevel   (numeric, optional, 0-4, default=3) How thorough the block verification is.
                check level 0: read from disk
                check level 1: verify block validity
                check level 2: verify undo validity
                check level 3: check for inconsistencies during memory-only disconnect of tip blocks
                check level 4: try reconnecting blocks
2. numblocks    (numeric, optional, default=288, 0=all) The number of blocks to check.
```

Result:

```
true|false       (boolean) Verified or not
```

Examples:

```
> verus verifychain
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "verifychain", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

verifytxoutproof "proof"

Verifies that a proof points to a transaction in a block, returning the transaction it commits to
and throwing an RPC error if the block is not in our best chain

Arguments:

```
1. "proof"    (string, required) The hex-encoded proof generated by gettxoutproof
```

Result:

```
["txid"]      (array, strings) The txid(s) which the proof commits to, or empty array if the proof is invalid
```

z_getbestblockhash

Return the hash of the best block.

Result:

```
"hash"    (string) The block hash
```

z_getblock "hash|height" verbosity

Return information about the given block.

Arguments:

1. "hash|height" (string, required) The block hash or height. Height can be negative where -1 is the last known valid block
2. verbosity (numeric, optional, default=1) 0 for hex encoded data, 1 for a json object, and 2 for json object with transaction data

Result:

```
{
  "hash" : "hash",          (string) The block hash
  "confirmations" : n,      (numeric) The number of confirmations, or -1 if the block is not on the best chain
  "size" : n,               (numeric) The size of the block
  "height" : n,             (numeric) The height of the block in the chain
  "version" : n,            (numeric) The block version
  "merkleroot" : "xxxx",    (string) The merkle root
  "finalsaplingroot" : "xxxx",   (string) The root of the Sapling commitment tree after applying all transactions in this block
  "tx" : [                  (array of string) The transaction ids
     "transactionid"        (string) The transaction id
     ,...
  ],
  "time" : ttt,             (numeric) The block time in seconds since epoch (Jan 1 1970 GMT)
  "nonce" : n,              (numeric) The nonce
  "solution" : "xxxx",      (string) The solution
  "bits" : "xxxxxxxx",      (string) The bits
  "difficulty" : x.xxx,     (numeric) The difficulty
  "chainwork" : "xxxx"      (string) Expected number of hashes required to produce the chain up to this block (in hex)
  "anchor" : "xxxx",        (string) The anchor for the block (last Sapling root)
  "chainSupply" : {
    "chainValue" : x.xxx,   (numeric) Total value held by transparent addresses in the chain
    "chainValueZ" : x.xxx,  (numeric) Total value held by sprout addresses in the chain
    "valueDelta" : x.xxx    (numeric) Change in value held by transparent addresses in this block
  },
  "valuePools" : [
    {
      "id" : "xxxx",        (string) Name of value pool
      "monitored" : xxxx,   (boolean) Whether or not this value pool is being tracked
      "chainValue" : x.xxx, (numeric) Total value held by this value pool in the chain
      "valueDelta" : x.xxx  (numeric) Change in value held by this value pool in this block
    },
    ...
  ],
  "previousblockhash" : "xxxx",  (string) The hash of the previous block
  "nextblockhash" : "xxxx"       (string) The hash of the next block
}
```

Examples:

```
> verus z_getbestblockhash "0000000fe5c173e1ac3e4f9197c943ca0f9e6c26ac241c2b26972d6fe4f287c9e"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_getbestblockhash", "params": ["0000000fe5c173e1ac3e4f9197c943ca0f9e6c26ac241c2b26972d6fe4f287c9e"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
> verus z_getbestblockhash 12000
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_getbestblockhash", "params": [12000] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

Control

getinfo

Returns an object containing various state info.

Result:

```
{
  "version": xxxxx,           (numeric) the server version
  "protocolversion": xxxxx,   (numeric) the protocol version
  "VRSCversion": "version",   (string) the komodo version
  "notarized": xxxxx,         (numeric) the most recent notarization block
  "prevMoMheight": xxxxx,     (numeric) previous notarization block
  "notarizedhash": "hash",    (hash) notarization hash
  "notarizedtxid": "hash",    (hash) notarization txid
  "notarizedtxid_height": "mm",
  "KMDnotarized_height": n,   (numeric)
  "notarized_confirms": n,    (numeric)
  "blocks": xxxxx,            (numeric) the current number of blocks processed in the server
  "timeoffset": xxxxx,        (numeric) the time offset
  "tiptime": xxxxx,           (numeric) the time in the tip of the chain
  "connections": xxxxx,       (numeric) the number of connections
  "proxy": "host:port",       (string, optional) the proxy used by the server
  "difficulty": xxxxxx,       (numeric) the current difficulty
  "testnet": true|false,      (boolean) if the server is using testnet or not

  "keypoololdest": xxxxxx,    (numeric) the timestamp (seconds since GMT epoch) of the oldest pre-generated key in the key pool
  "keypoolsize": xxxx,        (numeric) how many new keys are pre-generated
  "paytxfee": x.xxxx,         (numeric) the transaction fee set in VRSCTAB
  "tls_established": n,       (numeric) amount of TLS connections established
  "tls_verified": n,          (numeric) amount of verified TLS connections established
  "relayfee": x.xxxx,         (numeric) minimum relay fee for non-free transactions in VRSCTAB
  "errors": "..."             (string) any error messages
  "CCid": n,
  "name": "xxxxx",            (string) The name of chain
  "p2pport": xxxxx,           (numeric) number of the P2P port
  "rpcport": xxxxx,           (numeric) number of the RPC port
```

```
  "magic": xxxxx,            (numeric) chain magic number of the chain
  ...                        More chain definition specific parameters, which ones depends on the chain.
}
```

Examples:

```
> verus getinfo
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getinfo", "params": [] }'
-H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`help ( "command" )`

List all commands, or get help for a specified command.

Arguments:

```
1. "command"     (string, optional) The command to get help on
```

Result:

```
"text"     (string) The help text
```

`stop`

Stop Verus server.

Crosschain

`MoMoMdata symbol kmdheight ccid`

`assetchainproof needs a txid`

`calc_MoM height MoMdepth`

`getNotarisationsForBlock blockHash`

Takes a block hash and returns notarisation transactions within the block

`height_MoM height`

`migrate_completeimporttransaction importTx`

Takes a cross chain import tx with proof generated on assetchain and extends proof to target chain proof root

`migrate_converttoexport rawTx dest_symbol export_amount`

Convert a raw transaction to a cross-chain export.
If neccesary, the transaction should be funded using `fundrawtransaction`.
Finally, the transaction should be signed using `signrawtransaction`.
The finished export transaction, plus the payouts, should be passed to the "`migrate_createimporttransactio`n" method on a KMD node to get the corresponding import transaction.

`migrate_createimporttransaction burnTx payouts`

Create an importTx given a burnTx and the corresponding payouts, hex encoded

`scanNotarisationsDB blockHeight symbol [blocksLimit=1440]`

Scans notarisationsdb backwards from height for a notarisation of given symbol

Disclosure

`z_getpaymentdisclosure "txid" "js_index" "output_index" ("message")`

Generate a payment disclosure for a given joinsplit output.

EXPERIMENTAL FEATURE

> WARNING: z_getpaymentdisclosure is disabled.
> To enable it, restart verusd with the `-experimentalfeatures` and `-paymentdisclosure` commandline options, or add these two lines to the zcash.conf file:

```
experimentalfeatures=1
paymentdisclosure=1
```

Arguments:

```
1. "txid"           (string, required)
2. "js_index"       (string, required)
3. "output_index"   (string, required)
4. "message"        (string, optional)
```

Result:

```
"paymentdisclosure"  (string) Hex data string, with "zpd:" prefix.
```

Examples:

```
> verus z_getpaymentdisclosure 96f12882450429324d5f3b48630e3168220e49ab7b0f066e5c2935a6b88bb0f2 0 0 "refund"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_getpaymentdisclosure",
"params": ["96f12882450429324d5f3b48630e3168220e49ab7b0f066e5c2935a6b88bb0f2", 0, 0, "refund"] }' -H 'content-typ
e: text/plain;' http://127.0.0.1:27486/
```

`z_validatepaymentdisclosure "paymentdisclosure"`

Validates a payment disclosure.

EXPERIMENTAL FEATURE

> WARNING: z_validatepaymentdisclosure is disabled.
> To enable it, restart verusd with the `-experimentalfeatures` and
> `-paymentdisclosure` commandline options, or add these two lines
> to the zcash.conf file:

```
experimentalfeatures=1
paymentdisclosure=1
```

Arguments:

```
1. "paymentdisclosure"     (string, required) Hex data string, with "zpd:" prefix.
```

Examples:

```
> verus z_validatepaymentdisclosure "zpd:706462ff004c561a0447ba2ec51184e6c204..."
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_validatepaymentdisclosur
e", "params": ["zpd:706462ff004c561a0447ba2ec51184e6c204..."] }' -H 'content-type: text/plain;' http://127.0.0.1:
27486/
```

Generating

`generate numblocks`

Mine blocks immediately (before the RPC call returns)

> Note: this function can only be used on the regtest network

Arguments:

```
1. numblocks    (numeric) How many blocks are generated immediately.
```

Result:

```
[ blockhashes ]     (array) hashes of blocks generated
```

Examples:

Generate 11 blocks

```
> verus generate 11
```

`getgenerate`

Return if the server is set to mine and/or mint coins or not. The default is false.
It is set with the command line argument `-gen` (or `VRSC.conf` setting `gen`) and `-mint`
It can also be set with the `setgenerate` call.

Result

```
{
  "staking": true|false      (boolean) If staking is on or off (see setgenerate)
  "generate": true|false     (boolean) If mining is on or off (see setgenerate)
  "numthreads": n            (numeric) The processor limit for mining. (see setgenerate)
}
```

Examples:

```
> verus getgenerate
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getgenerate", "params": []
}' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`setgenerate generate ( genproclimit )`

Set 'generate' true to turn either mining/generation or minting/staking on and false to turn both off.
Mining is limited to 'genproclimit' processors, -1 is unlimited, setgenerate true with 0 genproclimit turns on staking
See the getgenerate call for the current setting.

Arguments:

```
1. generate         (boolean, required) Set to true to turn on generation, off to turn off.
2. genproclimit     (numeric, optional) Set processor limit when generation is on. Can be -1 for unlimited, 0 to
turn on staking.
```

Examples:

Set the generation on with a limit of one processor

```
> verus setgenerate true 1
```

Turn minting/staking on

```
> verus setgenerate true 0
```

Check the setting

```
> verus getgenerate
```

Turn off generation and minting

```
> verus setgenerate false
```

Using json rpc

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "setgenerate", "params": [t
rue, 1] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

Identity

`getidentitieswithaddress '{"address":"validprimaryaddress","fromheight":height, "toheight":height, "unspent":false}'`

Arguments:

```
{
    "address":"validaddress"     (string, required) returns all identities that contain the specified address in
its primary addresses
    "fromheight":n               (number, optional, default=0) Search for qualified identities modified from this
height forward only
    "toheight":n                 (number, optional, default=0) Search for qualified identities only up until this
height (0 == no limit)
    "unspent":bool               (bool, optional, default=false) if true, this will only return active ID UTXOs a
s of the current block height
}
```

Result:

```
[                                  (array) array of matching identities
  {identityobject},                (object) identity with additional member "txout" with txhash and output index
  ...
]
```

Examples:

```
> verus getidentitieswithaddress '{"address":"validprimaryaddress","fromheight":height, "toheight":height, "unspe
nt":false}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getidentitieswithaddress",
"params": ['{"address":"validprimaryaddress","fromheight":height, "toheight":height, "unspent":false}'] }' -H 'co
ntent-type: text/plain;' http://127.0.0.1:27486/
```

`getidentitieswithrecovery '{"identityid":"idori-address", "fromheight":height, "toheight":height, "unspent":false}'`

Arguments:

```
{
    "identityid":"idori-address" (string, required) returns all identities where this ID or i-address is the reco
very authority
    "fromheight":n               (number, optional, default=0) Search for qualified identities modified from this
height forward only
    "toheight":n                 (number, optional, default=0) Search for qualified identities only up until this
height (0 == no limit)
    "unspent":bool               (bool, optional, default=false) if true, this will only return active ID UTXOs a
s of the current block height
}
```

Result:

```
[                                  (array) array of matching identities
  {identityobject},                (object) identity with additional member "txout" with txhash and output index
  ...
]
```

Examples:

```
> verus getidentitieswithrecovery '{"identityid":"idori-address","fromheight":height,"toheight":height,"unspent":
false}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getidentitieswithrecover
y", "params": ['{"identityid":"idori-address","fromheight":height,"toheight":height,"unspent":false}'] }' -H 'con
tent-type: text/plain;' http://127.0.0.1:27486/
```

`getidentitieswithrevocation '{"identityid":"idori-address", "fromheight":height, "toheight":height, "unspent":false}'`

Arguments:

```
{
    "identityid":"idori-address" (string, required) returns all identities where this ID or i-address is the revo
cation authority
    "fromheight":n               (number, optional, default=0) Search for qualified identities modified from this
height forward only
    "toheight":n                 (number, optional, default=0) Search for qualified identities only up until this
height (0 == no limit)
    "unspent":bool               (bool, optional, default=false) if true, this will only return active ID UTXOs a
s of the current block height
}
```

Result:

```
[                                  (array) array of matching identities
  {identityobject},                (object) identity with additional member "txout" with txhash and output index
  ...
]
```

Examples:

```
> verus getidentitieswithrevocation '{"identityid":"idori-address","fromheight":height,"toheight":height,"unspen
t":false}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getidentitieswithrevocatio
n", "params": ['{"identityid":"idori-address","fromheight":height,"toheight":height,"unspent":false}'] }' -H 'con
tent-type: text/plain;' http://127.0.0.1:27486/
```

`getidentity "name@ || iid" (height) (txproof) (txproofheight)`

Arguments:

```
  "name@ || iid"                    (string, required) name followed by "@" or i-address of an identity
  "height"                          (number, optional) default=current height, return identity as of this heig
ht
  "txproof"                         (bool, optional) default=false, if true, returns proof of ID
  "txproofheight"                   (number, optional) default="height", height from which to generate a proof
```

Result:

```
{
  "identity": {
    "version": n,                 (integer) version number
    "flags": n,                   (integer) flag
    "primaryaddresses": [
      "xxxx"                      (text) Public address that controls this ID
    ],
    "minimumsignatures": n,       (integer) Amount of signatures required to spend UTXOs on this ID
    "identityaddress": "xxxx",    (text) i-address of the ID
    "parent": "xxxx",             (text) Parent ID of the chain the ID is on
    "name": "xxxx",               (text) The Identity Name
    "contentmap": {
    },
    "revocationauthority": "xxxx",  (text) the i-address of the identity that can revoke this ID
    "recoveryauthority": "xxxx",    (text) the i-address of the identity that can recover this ID when revoked
    "timelock": 0                 (integer) Timelock
  },
  "status": "xxxx",               (text) The revocation status ("Active"|"Revoked")
  "canspendfor": xxxx,            (boolean) Can the node running the command spend UTXOs on this ID (True|Fals
e)
  "cansignfor": xxxx,             (boolean) Can the node running the command sign for this ID (True|False)
  "blockheight": n,               (integer) blockheight the current ID info was put on the chain
  "txid": "xxxx",                 (string) Transaction ID at blockheight to put the current ID data on the chai
n
  "vout": n                       (integer) voutnumber of the TXID
}
```

Examples:

```
> verus getidentity "name@"
> verus getidentity "i5v3h9FWVdRFbNHU7DfcpGykQjRaHtMqu7"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getidentity", "params":
["name@"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

getidentitycontent "name@ || iid" (heightstart) (heightend) (txproofs) (txproofheight) (vdxfkey)

Arguments:

```
"name@ || iid"       (string, required) name followed by "@" or i-address of an identity
"heightstart"        (number, optional) default=0, only return content from this height forward, inclusive
"heightend"          (number, optional) default=0 which means max height, only return content up to this height,
                                        inclusive. -1 means also return values from the mempool.
"txproofs"           (bool, optional) default=false, if true, returns proof of ID
"txproofheight"      (number, optional) default="height", height from which to generate a proof
"vdxfkey"            (vdxf key, optional) default=null, more selective search for specific content in ID
```

Result:

Examples:

```
> verus getidentitycontent "name@"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getidentitycontent", "para
ms": ["name@"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

getidentityhistory "name@ || iid" (heightstart) (heightend) (txproofs) (txproofheight)

Arguments:

```
"name@ || iid"       (string, required) name followed by "@" or i-address of an identity
"heightstart"        (number, optional) default=0, only return content from this height forward, inclusive
"heightend"          (number, optional) default=0 which means max height, only return content up to this height,
                     inclusive. -1 means also return values from the mempool.
"txproofs"           (bool, optional) default=false, if true, returns proof of ID
"txproofheight"      (number, optional) default="height", height from which to generate a proof
```

Result:

Examples:

```
> verus getidentityhistory "name@"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getidentityhistory", "para
ms": ["name@"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

getidentitytrust '["id",...]'

Arguments:

```
"["id",...]"                                  (strarray, optional) if specified, only returns rating values
for specified IDs, otherwise all
```

Result:

```
{
  "setratings":{"id":JSONRatingObject,...},       (jsonobj) an ID/ratings key/value object
  "identitytrustmode":<n>                           (int) 0 = no restriction on sync, 1 = only sync to IDs rated
approved, 2 = sync to all IDs but those on block list
}
```

Examples:

```
> verus getidentitytrust '["id",...]'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getidentitytrust", "param
s": ['["id",...]'] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

listidentities (includecanspend) (includecansign) (includewatchonly)

Arguments:

```
"includecanspend"    (bool, optional, default=true)    Include identities for which we can spend/authorize
"includecansign"     (bool, optional, default=true)    Include identities that we can only sign for but not spend
"includewatchonly"   (bool, optional, default=false)   Include identities that we can neither sign nor spend, but
are either watched or are co-signers with us
```

Result:

```
[
  {
    "identity": {
      "version": n,                  (integer) version number
      "flags": n,                    (integer) flag
      "primaryaddresses": [
        "xxxx"                       (text) Public address that controls this ID
      ],
      "minimumsignatures": n,        (integer) Amount of signatures required to spend UTXOs on this ID
      "identityaddress": "xxxx",     (text) i-address of the ID
      "parent": "xxxx",              (text) Parent ID of the chain the ID is on
      "name": "xxxx",                (text) The Identity Name
      "contentmap": {
      },
      "revocationauthority": "xxxx", (text) the i-address of the identity that can revoke this ID
      "recoveryauthority": "xxxx",   (text) the i-address of the identity that can recover this ID when revoked
      "timelock": 0                  (integer) Timelock
    },
    "blockheight": n,                (integer) blockheight the current ID info was put on the chain
    "txid": "xxxx",                  (string) Transaction ID at blockheight to put the current ID data on the ch
ain
    "status": "xxxx",                (text) The revocation status ("Active"|"Revoked")
    "canspendfor": xxxx,             (boolean) Can the node running the command spend UTXOs on this ID (True|Fal
se)
    "cansignfor": xxxx,              (boolean) Can the node running the command sign for this ID (True|False)
  },
  ....
]
```

Examples:

```
> verus listidentities '{"name" : "myname"}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "listidentities", "params":
['{"name" : "myname"}'] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

recoveridentity "jsonidentity" (returntx) (tokenrecover) (feeoffer) (sourceoffunds)

Arguments:

```
      "returntx"                    (bool,   optional) defaults to false and transaction is sent, if true, t
ransaction is signed by this wallet and returned
      "tokenrevoke"                 (bool,   optional) defaults to false, if true, the tokenized ID control
token, if one exists, will be used to revoke
      "feeoffer"                    (value,  optional) non-standard fee amount to pay for the transaction
      "sourceoffunds"               (string, optional) transparent or private address to source all funds fo
r fees to preserve privacy of the identity
```

Result:

transactionid (hexstr)

Examples:

```
> verus recoveridentity '{"name" : "myname"}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "recoveridentity", "param
s": ['{"name" : "myname"}'] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

registeridentity "jsonidregistration" (returntx) feeoffer sourceoffunds

`registeridentity` needs the values from `RegisterNameCommitment` output to register a VerusID

Arguments:

```
{
    "txid": "hexid",              (hex, required)
                                          the transaction ID of the name committment for this ID name -
                                          Take fromRegisterNameCommitment's output - txid
    "namereservation": {
      "name": "namestr",          (string, required)
                                          the unique name in this commitment - Take from
                                          output - name
      "salt": "hexstr",           (hex, required)
                                          salt used to hide the commitment - Take from
                                          RegisterNameCommitment's output - salt
      "referral": "identityID",   (name@ or address, required)
                                          must be a valid ID to use as a referrer to receive a
                                          discount - Take from RegisterNameCommitment's
                                          output - referral
      "parent": "",               (name@ or address, required)
                                          must be a valid ID. This ID can be used to revoke and recover
                                          the nameID we regsiter with this current command - Take from
                                          RegisterNameCommitment's output - parent
      "nameid": "nameID"          (base58, required)
                                          identity address for this identity if it is created -
                                          Take from RegisterNameCommitment's output - nameid
    },
    "identity": {
      "name": "namestr",          (string, required)
                                          the unique name for this identity - Take from
                                          RegisterNameCommitment's output - name
      "parent": "IDParent",       (string, required for IDs with parents other than VRSC)
                        the parents friendly name e.g. gravity
      "primaryaddresses": [       (array of strings, required) the trasparent/public address(es)
        "hexstr"
      ],
      "minimumsignatures": 1,     (int, required)
                                          MofN signatures required out of the primary addresses list
                                          to sign transactions
      "privateaddress": "hexstr", (string, optional)
                                          shielded address associated with the VerusID being made
      "revocationauthority": "identityID", (name@ or address, optional)
                                          The ID entered here will be able to disable your created ID
                                          in case of loss or theft. It is some existing ID which either
                                          is under your own control or the ID you trust can help you
                                          revoke in case of this ID's theft.
      "recoveryauthority": "identityID"   (name@ or address, optional, required if "revocationauthority"
                                         is used)
                                          The ID entered here will be able to revive your created ID
                                          if it is revoked. It is some existing ID which either is
                                          under your own control or the ID you trust can help you
                                          revive in case of this ID's revoked.
    }
}
returntx                          (bool, optional) default=false if true, return a transaction for additional
                                          signatures rather than committing it
feeoffer                          (amount, optional)
                                          amount to offer miner/staker for the registration
                                          fee, if missing, uses standard price
sourceoffunds                     (addressorid, optional) optional address to use for source of funds. if not sp
ecified, transparent wildcard "*" is used
```

Result:

transactionid (hexstr)

Examples:

```
> verus registeridentity jsonidregistration
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "registeridentity", "param
s": [jsonidregistration] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

registernamecommitment "name" "controladdress" ("referralidentity") ("parentnameorid") ("sourceoffunds")

Registers a name commitment, which is required as a source for the name to be used when registering an identity. The name commitment hides the name itself while ensuring that the miner who mines in the registration cannot front-run the name unless they have also registered a name commitment for the same name or are willing to forfeit the offer of payment for the chance that a commitment made now will allow them to register the name in the future.

Arguments:

```
"name"                          (string, required)  the unique name to commit to. creating a name commitment is
not a
                                                     registration, and if one is created for a name that exist
s, it may
                                                     succeed, but will never be able to be used.
"controladdress"                (address, required) address that will control this commitment
"referralidentity"              (identity, optional)friendly name or identity address that is provided as a refe
rral mechanism and to lower network cost of the ID
"parentnameorid-pbaasonly"      (currency, optional)friendly name or currency i-address, which will be the paren
t of this ID and dictate issuance rules & pricing
"sourceoffunds"                 (addressorid, optional) optional address to use for source of funds. if not spec
ified, transparent wildcard "*" is used
```

Result: obj

```
{
    "txid" : "hexid"
    "namereservation" :
    {
        "name"    : "namestr",     (string) the unique name in this commitment
        "salt"    : "hexstr",      (hex)    salt used to hide the commitment
        "referral": "identityaddress", (base58) address of the referring identity if there is one
        "parent"  : "namestr",   (string) name of the parent if not Verus or Verus test
        "nameid"  : "address",   (base58) identity address for this identity if it is created
    }
}
```

Examples:

```
> verus registernamecommitment "name"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "registernamecommitment",
"params": ["name"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

revokeidentity "nameorID" (returntx) (tokenrevoke) (feeoffer) (sourceoffunds)

Arguments:

```
"nameorID"                      (string, required)  the unique name to commit to. creating a name commitment is
not a registration, and if one is
                                                     created for a name that exists, it may succeed, but will ne
ver be able to be used.
"returntx"                      (bool,   optional) defaults to false and transaction is sent, if true, transact
ion is signed by this wallet and returned
"tokenrevoke"                   (bool,   optional) defaults to false, if true, the tokenized ID control token,
if one exists, will be used to revoke
"feeoffer"                      (value,  optional) non-standard fee amount to pay for the transaction
"sourceoffunds"                 (string, optional) transparent or private address to source all funds for fees
to preserve privacy of the identity
```

Result:

```
{
    "txid" : "hexid"
    "namereservation" :
    {
        "name"    : "namestr",     (string) the unique name in this commitment
        "salt"    : "hexstr",      (hex)    salt used to hide the commitment
        "referral": "identityaddress", (base58) address of the referring identity if there is one
        "parent"  : "namestr",     (string) name of the parent if not Verus or Verus test
        "nameid"  : "address",     (base58) identity address for this identity if it is created
    }
}
```

Examples:

```
> verus revokeidentity "nameorID"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "revokeidentity", "params":
["nameorID"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

setidentitytimelock "id@" '{"unlockatblock":absoluteblockheight || "setunlockdelay":numberofblocksdelayafterunlock}' (returntx) (feeoffer) (sourceoffunds)

Enables timelocking and unlocking of funds access for an on-chain VerusID. This does not affect the lock status of VerusIDs on other chains,
including VerusIDs with the same identity as this one, which has been exported to another chain.

Use "setunlockdelay" to set a time unlock delay on an identity, which means that once the identity has been unlocked, numberofblocksdelayafterunlock must then pass before the identity will be able to spend funds on this blockchain. Services which support VerusID authentication and recognize this setting may also choose to prevent funds transfers when an ID is locked.

Use "unlockatblock" to either unlock, by passing the current block, which will still require waiting for the specified unlock delay, or to set a future unlock height that immediately begins counting down. Unlike an unlock delay, which only starts counting down when the ID is unlocked, an "unlockatblock" time lock is absolute and will automatically unlock when the specified block passes.

Arguments - either "unlockatblock" or "setunlockdelay" must be specified and not both

```
{
  "unlockatblock"               (number, optional) unlock at an absolute block height, countdown starts when min
ed into a block
  "setunlockdelay"              (number, optional) delay this many blocks after unlock request to unlock, can on
ly be
                                                    circumvented by revoke/recover
}
"returntx"                      (bool,   optional) defaults to false and transaction is sent, if true, transacti
on is signed by this wallet and returned
"feeoffer"                      (bool,   optional) non-standard fee amount to pay for the transaction
"sourceoffunds"                 (string,   optional) transparent or private address to source all funds for fees
to preserve privacy of the identity
```

Result:

Hex string of either the txid if returnhex is false or the hex serialized transaction if returntx is true.
If returntx is true, the transaction will not have been submitted and must be sent with "sendrawtransaction"
after any necessary signatures are applied in the case of multisig.

Examples:

```
> verus setidentitytimelock "id@" '{"unlockatblock":absoluteblockheight || "setunlockdelay":numberofblocksdelayaf
terunlock}' (returntx)
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "setidentitytimelock", "par
ams": ["id@" '{"unlockatblock":absoluteblockheight || "setunlockdelay":numberofblocksdelayafterunlock}' (returnt
x)] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

setidentitytrust '{"clearall": bool, "setratings":{"id":JSONRatingObject,...}, "removeratings":["id",...], "identitytrustmode":<n>}'

Arguments:

```
{
    "clearall": bool                          (bool, optional) clears all wallet identity trust lists before a
dding, removing, or trust mode operations
    "setratings":{"id":JSONRatingObject,...}  (obj, optional) replaces ratings for specified IDs with those gi
ven
    "removeratings":["id",...]                (strarray, optional) erases ratings for IDs specified
    "identitytrustmode": <n>                  (number, optional) 0 = no restriction on sync, 1 = only sync to
IDs rated approved, 2 = sync to all IDs but those on block list
}
```

Result

no return on success, else error

Examples

```
> verus setidentitytrust '{"clearall": bool, "setratings":{"id":JSONRatingObject,...}, "removeratings":["i
d",...], "identitytrustmode":<n>}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "setidentitytrust", "param
s": ['{"clearall": bool, "setratings":{"id":JSONRatingObject,...}, "removeratings":["id",...], "identitytrustmod
e":<n>}'] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

signdata 'json'

Full json:

```
{
```

```
  "address":"i-address or friendly name (t-address will result in simple signature w/indicated hash and prefix, n
othing else)",
  "prefixstring":"extra string that is hashed during signature and must be supplied for verification",
  "filename":"filepath/filename" |
  "message":"any message" |
  "messagehex":"hexdata" |
  "messagebase64":"base64data" |
  "datahash":"256bithex",
  "vdxfkeys":
    ["vdxfkey i-address", ...],
     "vdxfkeynames":["vdxfkeyname, object for getvdxfid API, or friendly name ID -- no i-addresses", ...],
     "boundhashes":["hexhash", ...],
  "hashtype": "sha256" | "sha256D" | "blake2b" | "keccak256"
  "signature":"currentsig"
}
```

Arguments:

```
{
  "address":"t-addr or identity"                            (string, required) The transparent address or iden
tity to use for signing.
  "filename" | "message" | "messagehex" | "messagebase64" | "datahash" (string, required) Data to sign
  "vdxfkeys":["vdxfkey", ...],                              (array, optional)  Array of vdxfkeys or ID i-addre
sses
  "vdxfkeynames":["vdxfkeyname", ...],                      (array, optional)  Array of vdxfkey names or fully
qualified friendly IDs
  "boundhashes":["hexhash", ...],                           (array, optional)  Array of bound hash values
  "hashtype"                                                (string, optional) one of: "sha256", "sha256D", "b
lake2b", "keccak256", defaults to sha256
  "signature"                                               (string, optional) The current signature of the me
ssage encoded in base 64 if multisig ID
}
```

Results:

```
{
  "hash":"hexhash"         (string) The hash of the message (SHA256, NOT SHA256D)
  "signature":"base64sig"  (string) The aggregate signature of the message encoded in base 64 if all or partial s
igning successful
}
```

Examples

Create the signature

```
> verus signdata '{"identity":"Verus Coin Foundation.vrsc@", "message":"hello world"}'
```

Verify the signature

```
> verus verifydata '{"identity":"Verus Coin Foundation.vrsc@", "message":"hello world", "signature":"base64sig"}'
```

As json rpc

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "signdata", "params": ['{"i
dentity":"Verus Coin Foundation.vrsc@", "message":"hello world"}'] }' -H 'content-type: text/plain;' http://127.
0.0.1:27486/
```

`signfile "address or identity" "filepath/filename" "curentsig"`

Generates a SHA256D hash of the file, returns the hash, and signs the hash with the private key specified

Arguments:

1. "t-addr or identity" (string, required) The transparent address or identity to use for signing.
2. "filename" (string, required) Local file to sign
3. "cursig" (string) The current signature of the message encoded in base 64 if multisig ID

Result:

```
{
  "hash":"hexhash"         (string) The hash of the message (SHA256, NOT SHA256D)
  "signature":"base64sig"  (string) The aggregate signature of the message encoded in base 64 if all or partial s
igning successful
}
```

Examples:

Create the signature

```
> verus signfile "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" "filepath/filename"
```

Verify the signature

```
> verus verifyfile "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" "signature" "filepath/filename"
```

As json rpc

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "signfile", "params": ["RD6
GgnrMpPaTSMn8vai6yiGA7mN4QGPV", "filepath/filename"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`signmessage "address or identity" "message" "curentsig"`

Sign a message with the private key of a t-addr or the authorities present in this wallet for an identity

Arguments:

1. "t-addr or identity" (string, required) The transparent address or identity to use for signing.
2. "message" (string, required) The message to create a signature of.
3. "cursig" (string) The current signature of the message encoded in base 64 if multisig ID

Result:

```
{
  "hash":"hexhash"         (string) The hash of the message (SHA256, NOT SHA256D)
  "signature":"base64sig"  (string) The aggregate signature of the message encoded in base 64 if all or partial s
igning successful
}
```

Examples:

Create the signature

```
> verus signmessage "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" "my message"
```

Verify the signature

```
> verus verifymessage "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" "signature" "my message"
```

As json rpc

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "signmessage", "params":
["RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV", "my message"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`updateidentity "jsonidentity" (returntx) (tokenupdate) (feeoffer) (sourceoffunds)`

Arguments:

```
{
      "version": n,                   (integer, optional) version number
      "flags": n,                     (integer, required) flag
      "primaryaddresses": [
        "xxxx", ...                   (text, required) Public addresses that control this ID
      ],
      "minimumsignatures": n,         (integer, required) Amount of signatures required to spend UTXOs on this ID
      "identityaddress": "xxxx",      (text, optional) i-address of the ID
      "parent": "xxxx",               (text, optional) Parent ID of the chain the ID is on
      "name": "xxxx",                 (text, required) the identity name
      "contentmap": {
      },
      "revocationauthority": "xxxx",  (text, optional) the i-address of the identity that can revoke this ID
      "recoveryauthority": "xxxx",    (text, optional) the i-address of the identity that can recover this ID whe
n revoked
      "timelock": 0,                  (integer, optional) timelock
            "privateaddress": "xxxx"        (text, optional) private address attached to the ID
}

feeoffer                        (amount, optional) amount to offer miner/staker for the registration fee, if m
issing, uses standard price
"returntx"                     (bool,   optional) defaults to false and transaction is sent, if true, transact
ion is signed by this wallet and returned
"tokenupdate"                  (bool,   optional) defaults to false, if true, the tokenized ID control token,
if one exists, will be used to update
                                                  which enables changing the revocation or recovery IDs, even
if the wallet holding the token does not
                                                  control either.
"feeoffer"                     (bool,   optional) non-standard fee amount to pay for the transaction
"sourceoffunds"                (string,   optional) transparent or private address to source all funds for fee
s to preserve privacy of the identity
```

Result:

transactionid (hexstr)

Examples:

```
> verus updateidentity '{"name" : "myname"}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "updateidentity", "params":
['{"name" : "myname"}'] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`verifyfile "address or identity" "signature" "filepath/filename" "checklatest"`

Verify a signed file

Arguments:

1. "t-addr or identity" (string, required) The transparent address or identity that signed the file.
2. "signature" (string, required) The signature provided by the signer in base 64 encoding (see signfile).
3. "filename" (string, required) The file, which must be available locally to the daemon and that was signed.
4. "checklatest" (bool, optional) If true, checks signature validity based on latest identity. defaults to false, which determines validity of signing height stored in signature.

Result:

true|false (boolean) If the signature is verified or not.

Examples:
Create the signature

```
> verus signfile "RNKiEBduBru6Siv1cZRVhp4fkZNyPska6z" "filepath/filename"
```

Verify the signature

```
> verus verifyfile "RNKiEBduBru6Siv1cZRVhp4fkZNyPska6z" "signature" "filepath/filename"
```

As json rpc

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "verifyfile", "params": ["R
NKiEBduBru6Siv1cZRVhp4fkZNyPska6z", "signature", "filepath/filename"] }' -H 'content-type: text/plain;' http://12
7.0.0.1:27486/
```

`verifyhash "address or identity" "signature" "hexhash" "checklatest"`

Verify a signed message

Arguments:

1. "t-addr or identity" (string, required) The transparent address or identity that signed the data.
2. "signature" (string, required) The signature provided by the signer in base 64 encoding (see signmessage/signfile).
3. "hexhash" (string, required) Hash of the message or file that was signed.
4. "checklatest" (bool, optional) If true, checks signature validity based on latest identity. defaults to false, which determines validity of signing height stored in signature.

Result:

true|false (boolean) If the signature is verified or not.

Examples:

Create the signature

```
> verus signfile "RNKiEBduBru6Siv1cZRVhp4fkZNyPska6z" "filepath/filename"
or
> verus signmessage "RNKiEBduBru6Siv1cZRVhp4fkZNyPska6z" "my message"
```

Verify the signature

```
> verus verifyhash "RNKiEBduBru6Siv1cZRVhp4fkZNyPska6z" "signature" "hexhash"
```

As json rpc

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "verifyhash", "params": ["R
NKiEBduBru6Siv1cZRVhp4fkZNyPska6z", "signature", "hexhash"] }' -H 'content-type: text/plain;' http://127.0.0.1:27
486/
```

`verifymessage "address or identity" "signature" "message" "checklatest"`

Verify a signed message

Arguments:

1. "t-addr or identity" (string, required) The transparent address or identity that signed the message.
2. "signature" (string, required) The signature provided by the signer in base 64 encoding (see signmessage).
3. "message" (string, required) The message that was signed.
4. "checklatest" (bool, optional) If true, checks signature validity based on latest identity. defaults to false, which determines validity of signing height stored in signature.

Result:

true|false (boolean) If the signature is verified or not.

Examples:

Create the signature

```
> verus signmessage "RNKiEBduBru6Siv1cZRVhp4fkZNyPska6z" "my message"
```

Verify the signature

```
> verus verifymessage "RNKiEBduBru6Siv1cZRVhp4fkZNyPska6z" "signature" "my message"
```

As json rpc

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "verifymessage", "params":
["RNKiEBduBru6Siv1cZRVhp4fkZNyPska6z", "signature", "my message"] }' -H 'content-type: text/plain;' http://127.0.
0.1:27486/
```

`Verifysignature 'json'`

Checks to see if the signature is valid and returns an error for invalid

Arguments

```
{
  "address":"t-addr or identity"                            (string, required) The transparent address or iden
tity to verify against the signature
                                                            (t-address checks on simple signature w/hash and p
refix, nothing else)
  "prefixstring":"extra string that is hashed during signature and must be supplied for verification",
  "filename" | "message" | "messagehex" | "messagebase64" | "datahash" (string, required) Data or hash of data si
gned
  "vdxfkeys":["vdxfkey", ...],                              (array, optional)  Array of vdxfkeys or ID i-addre
sses
  "vdxfkeynames":["vdxfkeyname", ...],                      (array, optional)  Array of objects for getvdxfid
API, or friendly name ID -- no i-addresses
  "boundhashes":["hexhash", ...],                           (array, optional)  Array of bound hash values
  "hashtype"                                                (string, optional) one of: "sha256", "sha256D", "b
lake2b", "keccak256", defaults to sha256
  "signature"                                               (string, optional) The current signature of the me
ssage encoded in base 64
  "checklatest"                                             (bool, optional)   If true, checks signature valid
ity based on latest identity. defaults to false,
                                                            which determines validity of signing height stored
in signature.
}
```

Results:

```
{
  "hash":"hexhash"         (string) The hash of the message (SHA256, NOT SHA256D)
  "signature":"base64sig"  (string) The aggregate signature of the message encoded in base 64 if all or partial s
igning successful
}
```

Examples:

```
> verus verifysignature '{"identity":"Verus Coin Foundation.vrsc@", "message":"hello world", "signature":"base64s
ig"}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "verifysignature", "param
s": ['{"identity":"Verus Coin Foundation.vrsc@", "message":"hello world", "signature":"base64sig"}'] }' -H 'conte
nt-type: text/plain;' http://127.0.0.1:27486/
```

Marketplace

`closeoffers ('["offer1_txid", "offer2_txid", ...]') (transparentorprivatefundsdestination) (privatefundsdestination)`

Closes all offers listed, if they are still valid and belong to this wallet.
Always closes expired offers, even if no parameters are given

Arguments

1. `["offer1_txid", "offer2_txid", ...]` (array, optional) array of hex tx ids of offers to close
2. transparentorprivatefundsdestination (transparent or private address, optional) destination for closing funds
3. privatefundsdestination (private address, optional) destination for native funds only

Results

null return

`getoffers "currencyorid" (iscurrency) (withtx)`

Returns all open offers for a specific currency or ID

Arguments

1. "currencyorid" (string, required) The currency or ID to check for offers, both sale and purchase
2. "iscurrency" (bool, optional) default=false, if false, this looks for ID offers, if true, currencies
3. "withtx" (bool, optional) default=false, if true, this returns serialized hex of the exchange transaction for signing

Result:

all available offers for or in the indicated currency or ID are displayed

Examples:

```
 verus getoffers "currencyorid" (iscurrency)
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getoffers", "params": ["cu
rrencyorid" (iscurrency)] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`listopenoffers (unexpired) (expired)'`

Shows offers outstanding in this wallet

Arguments

1. unexpired (bool, optional) default=true, list those offers in the wallet which are not expired
2. expired (bool, optional) default=true, list those offers in the wallet which are expired

Result

All open offers

`makeoffer fromaddress '{"changeaddress":"transparentoriaddress", "expiryheight":blockheight, "offer":{"currency":"anycurrency", "amount":...} | {"identity":"idnameoriaddress",...}', "for":{"address":..., "currency":"anycurrency", "amount":...} | {"name":`

This sends a transaction which provides a completely decentralized, fully on-chain an atomic swap offer for
"decentralized swapping of any blockchain asset, including any/multi currencies, NFTs, identities, contractual
"agreements and rights transfers, or to be used as bids for an on-chain auction of any blockchain asset(s).
"Sources and destination of funds for swaps can be any valid transparent address capable of holding or controlling
the specific asset.

Arguments

1. "fromaddress" (string, required) The VerusID, or wildcard address to send funds from. "*", "R*", or "i*" are valid wildcards
2. {
   "changeaddress" (string, required) Change destination when constructing transactions
   "offer" (object, required) Funds description or identity name, "address" in this object should be an address of the person making an offer for change
   "for" (object, required) Funds description or full identity description
   }
3. "returntx" (bool, optional) default = false, if true, returns a transaction waiting for taker completion instead of posting
4. "feeamount" (value, optional) default = 0.0001

Result:

```
{
  "txid" : "transactionid", The hex transaction id on success
  "hex" : "serializedtx"   If hex is requested, hex serialization of partial transaction instead of txid is retur
ned on success
}
```

Examples:

```
> verus makeoffer fromaddress '{"changeaddress":"transparentoriaddress", "expiryheight":blockheight, "offer":{"cu
rrency":"anycurrency", "amount":...} | {"identity":"idnameoriaddress",...}', "for":{"address":..., "currency":"an
ycurrency", "amount":...} | {"name":"identityforswap","parent":"parentid","primaryaddresses":["R-address(s)"],"mi
nimumsignatures":1,...}}' (returntx) (feeamount)
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "makeoffer", "params": [fro
maddress '{"changeaddress":"transparentoriaddress", "expiryheight":blockheight, "offer":{"currency":"anycurrenc
y", "amount":...} | {"identity":"idnameoriaddress",...}', "for":{"address":..., "currency":"anycurrency", "amoun
t":...} | {"name":"identityforswap","parent":"parentid","primaryaddresses":["R-address(s)"],"minimumsignatures":
1,...}}' (returntx) (feeamount)] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`takeoffer fromaddress '{"txid":"txid" | "tx":"hextx", "changeaddress":"transparentoriaddress", "deliver":"fullidnameoriaddresstodeliver" | {"currency":"currencynameorid","amount":n}, "accept":{"address":"addressorid","currency":"currencynameorid","amount`

If the current wallet can afford the swap, this accepts a swap offer on the blockchain, creates a transaction
to execute it, and posts the transaction to the blockchain.

Arguments:

1. "fromaddress" (string, required) The Sapling, VerusID, or wildcard address to send funds from, including fees for ID swaps.
   "*", "R*", or "i*" are valid wildcards
2. {
   "txid" (string, required) The transaction ID for the offer to accept
   "tx" (string, required) The hex transaction to complete in order to accept the offer
   "deliver" (object, required) One of "fullidnameoriaddresstotrade" or {"currency":"currencynameorid", "amount":value}
   "feeamount" (number, optional) Specific fee amount requested instead of default miner's fee
   }
3. "returntx" (bool, optional) default = false, if true, returns a transaction waiting for taker completion instead of posting
4. "feeamount" (value, optional) default = 0.0001

Result:

"txid" : "transactionid" (string) The transaction id if (returntx) is false
"hextx" : "hex" (string) The hexadecimal, serialized transaction if (returntx) is true

Examples:

```
> verus takeoffer fromaddress '{"txid":"txid" | "tx":"hextx", "deliver":"fullidnameoriaddresstodeliver" | {"curre
ncy":"currencynameorid","amount"}, "accept":{"address":"addressorid","currency":"currencynameorid","amount"} | {i
dentitydefinition}}' (returntx) (feeamount)
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "takeoffer", "params": [fro
maddress {"txid":"txid" | "tx":"hextx", "deliver":"fullidnameoriaddresstodeliver" | {"currency":"currencynameori
d","amount"}, "accept":{"address":"addressorid","currency":"currencynameorid","amount"} | {identitydefinition}}
(returntx) (feeamount)] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

Mining

`getblocksubsidy height`

Returns block subsidy reward, taking into account the mining slow start and the founders reward, of block at index provided.

Arguments:

1. height (numeric, optional) The block height. If not provided, defaults to the current height of the chain.

Result:

```
{
  "miner" : x.xxx           (numeric) The coinbase reward amount in VRSC.
}
```

Examples:

```
> verus getblocksubsidy 1000
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getblockubsidy", "params":
[1000] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getblocktemplate ( "jsonrequestobject" )`

If the request parameters include a 'mode' key, that is used to explicitly select between the default 'template' request or a 'proposal'.
It returns data needed to construct a block to work on.
See https://en.bitcoin.it/wiki/BIP_0022 for full specification.

Arguments:

1. "jsonrequestobject" (string, optional) A json object in the following spec

```
    {
      "mode":"template"    (string, optional) This must be set to "template" or omitted
      "rewarddistribution":{
          "(recipientaddress)":n,  (addressorid, relativeweight) key value to determine distribution
          "(recipientaddress)":n,
          "...
      "},
      "capabilities":[       (array, optional) A list of strings
          "support"           (string) client side supported feature, 'longpoll', 'coinbasetxn', 'coinbasevalu
e', 'proposal', 'serverlist', 'workid'
          ,...
        ]
    }
```

Result:

```
{
  "version" : n,                    (numeric) The block version
  "previousblockhash" : "xxxx",    (string) The hash of current highest block
  "finalsaplingroothash" : "xxxx", (string) The hash of the final sapling root
  "transactions" : [                (array) contents of non-coinbase transactions that should be included in the
next block
      {
         "data" : "xxxx",          (string) transaction data encoded in hexadecimal (byte-for-byte)
         "hash" : "xxxx",          (string) hash/id encoded in little-endian hexadecimal
         "depends" : [              (array) array of numbers
             n                      (numeric) transactions before this one (by 1-based index in 'transactions'
list) that must be present in the final block if this one is
             ,...
         ],
         "fee": n,                  (numeric) difference in value between transaction inputs and outputs (in Sat
oshis); for coinbase transactions, this is a negative Number of the total collected block fees (ie, not including
the block subsidy); if key is not present, fee is unknown and clients MUST NOT assume there isn't one
         "sigops" : n,              (numeric) total number of SigOps, as counted for purposes of block limits; i
f key is not present, sigop count is unknown and clients MUST NOT assume there aren't any
         "required" : true|false     (boolean) if provided and true, this transaction must be in the final block
      }
      ,...
  ],
  "coinbasetxn" : { ... },          (json object) information for coinbase transaction
  "target" : "xxxx",               (string) The hash target
  "mintime" : xxx,                  (numeric) The minimum timestamp appropriate for next block time in seconds s
ince epoch (Jan 1 1970 GMT)
  "mutable" : [                     (array of string) list of ways the block template may be changed
     "value"                        (string) A way the block template may be changed, e.g. 'time', 'transaction
s', 'prevblock'
     ,...
  ],
  "noncerange" : "00000000ffffffff",   (string) A range of valid nonces
  "sigoplimit" : n,                (numeric) limit of sigops in blocks
  "sizelimit" : n,                 (numeric) limit of block size
  "curtime" : ttt,                 (numeric) current timestamp in seconds since epoch (Jan 1 1970 GMT)
  "bits" : "xxx",                 (string) compressed target of next block
  "height" : n                     (numeric) The height of the next block
}
```

Examples:

```
> verus getblocktemplate
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getblocktemplate", "param
s": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getlocalsolps`

Returns the average local solutions per second since this node was started.
This is the same information shown on the metrics screen (if enabled).

Result:

xxx.xxxxx (numeric) Solutions per second average

Examples:

```
> verus getlocalsolps
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getlocalsolps", "params":
[] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getminingdistribution`

Retrieves current mining distribution

Arguments

NONE

Result:

```
NULL object if not set
If set:
{
  "uniquedestination1":value    (key/number) valid destination address and relative value output to it
  "uniquedestination2":value    (key/number) destination address and relative value output
  ...
}
```

Examples:

```
> verus getminingdistribution
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getminingdistribution", "p
arams": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getmininginfo`

Returns a json object containing mining-related information.

Result:

```
{
  "blocks": nnn,             (numeric) The current block
  "currentblocksize": nnn,   (numeric) The last block size
  "currentblocktx": nnn,     (numeric) The last block transaction
  "averageblockfees": xxx.xxxxx (numeric) The average block fees, in addition to block reward, over the past 100
blocks
  "difficulty": xxx.xxxxx    (numeric) The current difficulty
  "stakingsupply": xxx.xxxxx (numeric) The current estimated total staking supply
  "errors": "..."          (string) Current errors
  "generate": true|false     (boolean) If the generation is on or off (see getgenerate or setgenerate calls)
  "genproclimit": n          (numeric) The processor limit for generation. -1 if no generation. (see getgenerate
or setgenerate calls)
  "localsolps": xxx.xxxxx    (numeric) The average local solution rate in Sol/s since this node was started
  "networksolps": x          (numeric) The estimated network solution rate in Sol/s
  "pooledtx": n              (numeric) The size of the mem pool
  "testnet": true|false      (boolean) If using testnet or not
  "chain": "xxxx",         (string) current network name as defined in BIP70 (main, test, regtest)
  "generate": true|false     (boolean) If this instance is mining or staking
  "staking": true|false      (boolean) If staking
  "numthreads": n            (numeric) Number of CPU threads mining
  "mergemining": n           (numeric) Number of blockchains we are merge mining with
  "mergeminedchains": []     (optional, list of names) Blockchain names that are being merge mined with this bloc
kchain
}
```

Examples:

```
> verus getmininginfo
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getmininginfo", "params":
[] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getnetworkhashps ( blocks height )`

DEPRECATED - left for backwards-compatibility. Use getnetworksolps instead.

Returns the estimated network solutions per second based on the last n blocks.
Pass in [blocks] to override # of blocks, -1 specifies over difficulty averaging window.
Pass in [height] to estimate the network speed at the time when a certain block was found.

Arguments:

```
1. blocks     (numeric, optional, default=120) The number of blocks, or -1 for blocks over difficulty averaging w
indow.
2. height     (numeric, optional, default=-1) To estimate at the time of the given height.
```

Result:

```
x            (numeric) Solutions per second estimated
```

Examples:

```
> verus getnetworkhashps
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getnetworkhashps", "param
s": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getnetworksolps ( blocks height )`

Returns the estimated network solutions per second based on the last n blocks.
Pass in [blocks] to override # of blocks, -1 specifies over difficulty averaging window.
Pass in [height] to estimate the network speed at the time when a certain block was found.

Arguments:

```
1. blocks     (numeric, optional, default=120) The number of blocks, or -1 for blocks over difficulty averaging w
indow.
2. height     (numeric, optional, default=-1) To estimate at the time of the given height.
```

Result:

```
x            (numeric) Solutions per second estimated
```

Examples:

```
> verus getnetworksolps
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getnetworksolps", "param
s": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`prioritisetransaction <txid> <priority delta> <fee delta>`

Accepts the transaction into mined blocks at a higher (or lower) priority

Arguments:

```
1. "txid"         (string, required) The transaction id.
2. priority delta (numeric, required) The priority to add or subtract.
                  The transaction selection algorithm considers the tx as it would have a higher priority.
                  (priority of a transaction is calculated: coinage * value_in_satoshis / txsize)
3. fee delta      (numeric, required) The fee value (in satoshis) to add (or subtract, if negative).
                  The fee is not actually paid, only the algorithm for selecting transactions into a block
                  considers the transaction as it would have paid a higher (or lower) fee.
```

Results:

```
true              (boolean) Returns true
```

Examples:

```
> verus prioritisetransaction "txid" 0.0 10000
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "prioritisetransaction", "p
arams": ["txid", 0.0, 10000] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`setminingdistribution ( "jsonminingdistribution" )`

Sets multiple mining outputs with amounts that will be used to calculate relative outputs to each address for any reward

Arguments

```
{
  "uniquedestination1":value    (key/number, required) valid destination address and relative value output to it
  "uniquedestination2":value    (key/number, optional) destination address and relative value output
  ...
}
```

Result

NULL for success, exception otherwise

Examples:

```
> verus setminingdistribution {"myaddress":0.5, "otheraddress":0.5}
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "setminingdistribution", "p
arams": [{"myaddress":0.5, "otheraddress":0.5}] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`submitblock "hexdata" ( "jsonparametersobject" )`

Attempts to submit new block to network.
The 'jsonparametersobject' parameter is currently ignored.
See https://en.bitcoin.it/wiki/BIP_0022 for full specification.

Arguments:

```
1. "hexdata"    (string, required) the hex-encoded block data to submit
2. "jsonparametersobject"     (string, optional) object of optional parameters
    {
      "workid" : "id"    (string, optional) if the server provided a workid, it MUST be included with submissions
    }
```

Result:

```
"duplicate" - node already has valid copy of block
"duplicate-invalid" - node already has block, but it is invalid
"duplicate-inconclusive" - node already has block but has not validated it
"inconclusive" - node has not validated the block, it may not be on the node's current best chain
"rejected" - block was rejected as invalid
For more information on submitblock parameters and results, see: https://github.com/bitcoin/bips/blob/master/bip-
0022.mediawiki#block-submission
```

Examples:

```
> verus submitblock "mydata"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "submitblock", "params":
["mydata"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

Multichain

`addmergedblock "hexdata" ( "jsonparametersobject" )`

Adds a fully prepared block and its header to the current merge mining queue of this daemon.
Parameters determine the action to take if adding this block would exceed the available merge mining slots.
Default action to take if adding would exceed available space is to replace the choice with the least ROI if this block provides more.

Arguments:

```
1. "hexdata"                    (string, required) the hex-encoded, complete, unsolved block data to add. nTime,
and nSolution are replaced.
2. "name"                       (string, required) chain name symbol
3. "rpchost"                    (string, required) host address for RPC connection
4. "rpcport"                    (int,    required) port address for RPC connection
5. "userpass"                   (string, required) credentials for login to RPC
```

Result:

```
"deserialize-invalid" - block could not be deserialized and was rejected as invalid
"blocksfull"          - block did not exceed others in estimated ROI, and there was no room for an additional mer
ge mined block
```

Examples:

```
> verus addmergedblock "hexdata" '{"currencyid" : "hexstring", "rpchost" : "127.0.0.1", "rpcport" : portnum}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "addmergedblock", "params":
["hexdata" '{"currencyid" : "hexstring", "rpchost" : "127.0.0.1", "rpcport" : portnum, "estimatedroi" : (verusrew
ard/hashrate)}'] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`definecurrency '{"name": "coinortokenname", ..., "nodes":[{"networkaddress":"identity"},..]}' '({"name": "fractionalgatewayname", ..., })' ({"name": "reserveonename", ..., }) ...`

This defines a blockchain currency, either as an independent blockchain, or as a token on this blockchain. It also spends the identity after which this currency is named and sets a bit indicating that it has a currently active blockchain in its name.

To create a currency of any kind, the identity it is named after must be minted on the blockchain on which the currency is created.
Once a currency is activated for an identity name, the same symbol may not be reused for another currency or blockchain, even if the identity is transferred, revoked or recovered, unless there is an endblock specified and the currency or blockchain has deactivated as of that end block.

All funds to start the currency and for initial conversion amounts must be available to spend from the identity with the same name and ID as the currency being defined.

Arguments:

```
{
   "options" : n,                 (int,    optional) bits (in hexadecimal):
                                             OPTION_FRACTIONAL = 1                 // (1 decimal )allows reserve co
nversion using base calculations when set
                                             OPTION_ID_ISSUANCE = 2                // (2 decimal) clear is permissi
onless, if set, IDs may only be created by controlling ID
                                             OPTION_ID_STAKING = 4                 // (4 decimal) all IDs on chain
stake equally, rather than value-based staking
                                             OPTION_ID_REFERRALS = 8               // (8 decimal) if set, this chai
n supports referrals
                                             OPTION_ID_REFERRALREQUIRED = 0x10     // (16 decimal) if set, this cha
in requires referrals
                                             OPTION_TOKEN = 0x20                   // (32 decimal) if set, this is
a token, not a native currency
                                             OPTION_SINGLECURRENCY = 0x40          // (64 decimal) for PBaaS chains
or gateways to potentially restrict to single currency
                                             OPTION_GATEWAY = 0x80                 // (128 decimal) if set, this ro
utes external currencies
                                             OPTION_PBAAS = 0x100                  // (256 decimal) this is a PBaaS
chain definition
                                             OPTION_GATEWAY_CONVERTER = 0x200      // (512 decimal) this means that
for a specific PBaaS gateway, this is the default converter and will publish prices
                                             OPTION_GATEWAY_NAMECONTROLLER = 0x400  // (1024 decimal) when not set o
n a gateway, top level ID and currency registration happen on launch chain
                                             OPTION_NFT_TOKEN = 0x800              // (2048 decimal) single satoshi
NFT token, tokenizes control over the root ID
                                             OPTIONS_FLAG_MASK = 0xfff
  "name" : "xxxx",                (string, required) name of existing identity with no active or pending blockch
ain
  "idregistrationfees" : "xx.xx",  (value, required) price of an identity in native currency
  "idreferrallevels" : n,         (int, required) how many levels ID referrals go back in reward
  "notaries" : "[identity,..]",    (list, optional) list of identities that are assigned as chain notaries
  "minnotariesconfirm" : n,        (int, optional) unique notary signatures required to confirm an auto-notarizat
ion
  "notarizationreward" : "xx.xx",  (value,  required) default VRSC notarization reward total for first billing pe
riod
  "billingperiod" : n,            (int,    optional) number of blocks in each billing period
  "proofprotocol" : n,            (int,    optional) if 2, currency can be minted by whoever controls the ID
  "startblock"    : n,            (int,    optional) VRSC block must be notarized into block 1 of PBaaS chain, d
efault curheight + 100
  "endblock"      : n,            (int,    optional) chain is considered inactive after this block height, and a
new one may be started
  "currencies"    : "["VRSC",..]", (list, optional) reserve currencies backing this chain in equal amounts
  "weights"       : "["xx.xx",..]",(list, optional) the weight of each reserve currency in a fractional currency
  "conversions"   : "["xx.xx",..]",(list, optional) if present, must be same size as currencies. pre-launch conve
rsion ratio overrides
  "minpreconversion" : "["xx.xx",..]", (list, optional) must be same size as currencies. minimum in each currency
to launch
  "maxpreconversion" : "["xx.xx",..]", (list, optional) maximum in each currency allowed
  "initialcontributions" : "["xx.xx",..]", (list, optional) initial contribution in each currency
  "prelaunchdiscount" : "xx.xx"   (value, optional) for fractional reserve currencies less than 100%, discount on
final price at launch
  "initialsupply" : "xx.xx"       (value, required for fractional) supply after conversion of contributions, befo
re preallocation
  "prelaunchcarveout" : "0.xx",   (value, optional) identities and % of pre-converted amounts from each reserve c
urrency
  "preallocations" : "[{"identity":xx.xx}..]", (list, optional)  list of identities and amounts from pre-allocati
on
  "gatewayconvertername" : "name", (string, optional) if this is a PBaaS chain, this names a co-launched gateway
converter currency
  "blocktime"         : n,        (int, optional) target time in seconds to average between blocks (default 60 se
conds)
  "powaveragingwindow" : n,       (int, optional) total number of blocks to look back when averaging for DAA (def
ault 45 blocks)
  "notarizationperiod" : n,       (int, optional) min period miners/stakers must wait to post new notarization to
chain (default 10 min at any blocktime)
  "eras"          : "objarray",   (array, optional) data specific to each era, maximum 3
  {
   "reward"     : n,              (int64,  required) native initial block rewards in each period
   "decay"      : n,              (int64,  optional) reward decay for each era
   "halving"    : n,              (int,    optional) halving period for each era
   "eraend"     : n,              (int,    optional) ending block of each era
}
"nodes"         : "[obj, ..]", (objectarray, optional) up to 5 nodes that can be used to connect to the blockchai
n
 [{
   "networkaddress" : "ip:port",  (string,  optional) internet, TOR, or other supported address for node
   "nodeidentity" : "name@",      (string, optional) published node identity
  }, .. ]
}
```

Result:

```
{
  "txid" : "transactionid",       (string) The transaction id
  "tx"   : "json",                (json)   The transaction decoded as a transaction
  "hex"  : "data"                 (string) Raw data for signed transaction
}
```

Examples:

```
> verus definecurrency jsondefinition
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "definecurrency", "params":
[jsondefinition] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`estimateconversion '{"currency":"name","convertto":"name","amount":n}'`

This estimates conversion from one currency to another, taking into account pending conversions, fees and slippage.

Arguments

```
{
  "currency": "name"       (string, required)  Name of the source currency to send in this output, defaults to
                                               native of chain
  "amount":amount          (numeric, required) The numeric amount of currency, denominated in source currency
  "convertto":"name",      (string, optional)  Valid currency to convert to, either a reserve of a fractional, or
fractional
  "preconvert":"false",    (bool,  optional)   Convert to currency at market price (default=false), only works if
                                               transaction is mined before start of currency
  "via":"name",            (string, optional)  If source and destination currency are reserves, via is a common f
ractional

                                               to convert through
}
```

Result

```
{
  "estimatedcurrencystate": object              Estimation of all currency values, including prices and changes
}
```

Examples:

```
> verus estimateconversion '{"currency":"name","convertto":"name","amount":n}'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "estimateconversion", "para
ms": ['{"currency":"name","convertto":"name","amount":n}'] }' -H 'content-type: text/plain;' http://127.0.0.1:274
86/
```

`getbestproofroot '{"proofroots":["version":n,"type":n,"systemid":"currencyidorname","height":n, "stateroot":"hex","blockhash":"hex","power":"hex"],"lastconfirmed":n}'`

Determines and returns the index of the best (most recent, valid, qualified) proof root in the list of proof roots,
and the most recent, valid proof root.

Arguments

```
{
  "proofroots":                 (array, required/may be empty) ordered array of proof roots, indexed on return
  [
    {
      "version":n               (int, required) version of this proof root data structure
      "type":n                  (int, required) type of proof root (chain or system specific)
      "systemid":"hexstr"       (hexstr, required) system the proof root is for
      "height":n                (uint32_t, required) height of this proof root
      "stateroot":"hexstr"      (hexstr, required) Merkle or merkle-style tree root for the specified block/sequ
ence
      "blockhash":"hexstr"      (hexstr, required) hash identifier for the specified block/sequence
      "power":"hexstr"          (hexstr, required) work, stake, or combination of the two for most-work/most-pow
er rule
    }
  .
  .
  .
  ]
  "currencies":["id1"]        (array, optional) currencies to query for currency states
  "lastconfirmed":n             (int, required) index into the proof root array indicating the last confirmed ro
ot)
```

Result:

"bestindex" (int) index of best proof root not confirmed that is provided, confirmed index, or -1"latestproofroot"
(object) latest valid proof root of chain"currencystates"
(int) currency states of target currency and published bridges
Examples:

```
> verus getbestproofroot "{"proofroots":["version":n,"type":n,"systemid":"currencyidorname","height":n,"stateroo
t":"hex","blockhash":"hex","power":"hex"],"lastconfirmed":n}"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getbestproofroot", "param
s": ["{"proofroots":["version":n,"type":n,"systemid":"currencyidorname","height":n,"stateroot":"hex","blockhas
h":"hex","power":"hex"],"lastconfirmed":n}"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getcurrency "currencyname"`

Returns a complete definition for any given chain if it is registered on the blockchain. If the chain requested is NULL, chain definition of the current chain is returned.

Arguments:

```
1. "currencyname"               (string, optional) name of the chain to look for. no parameter returns curr
ent chain in daemon.
```

Result:

```
{
  "version" : n,                      (int) version of this chain definition
  "name" : "string",                  (string) name or symbol of the chain, same as passed
  "fullyqualifiedname" : "string",    (string) name or symbol of the chain with all parent namespaces, separat
ed by "."
  "currencyid" : "i-address",         (string) string that represents the currency ID, same as the ID behind t
he currency
  "currencyidhex" : "hex",            (string) hex representation of currency ID, getcurrency API supports "he
x:currencyidhex"
  "parent" : "i-address",             (string) parent blockchain ID
  "systemid" : "i-address",           (string) system on which this currency is considered to run
  "launchsystemid" : "i-address",     (string) system from which this currency was launched
  "notarizationprotocol" : n          (int) protocol number that determines variations in cross-chain or bridg
ed notarizations
  "proofprotocol" : n                 (int) protocol number that determines variations in cross-chain or bridg
ed proofs
  "startblock" : n,                   (int) block # on this chain, which must be notarized into block one of t
he chain
  "endblock" : n,                     (int) block # after which, this chain's useful life is considered to be
over
  "currencies" : "["i-address", ...]",  (stringarray) currencies that can be converted to this currency at launc
h or makeup a liquidity basket
  "weights" : "[n, ...]",             (numberarray) relative currency weights (only returned for a liquidity b
asket)
  "conversions" : "[n, ...]",         (numberarray) pre-launch conversion rates for non-fractional currencies
  "minpreconversion" : "[n, ...]",    (numberarray) minimum amounts required in pre-conversions for currency t
o launch
  "currencies" : "["i-address", ...]",  (stringarray) currencies that can be converted to this currency at launc
h or makeup a liquidity basket
  "currencynames" : "{"i-address":"fullname",...}", (obj) i-addresses mapped to fully qualified names of all sub-
currencies
  "initialsupply" : n,                (number) initial currency supply for fractional currencies before preall
ocation or issuance
  "prelaunchcarveout" : n,            (number) pre-launch percentage of proceeds for fractional currency sent
to launching ID
  "preallocations" : "[{"i-address":n}, ...]", (objarray) VerusIDs and amounts for pre-allocation at launch
  "initialcontributions" : "[n, ...]",  (numberarray) amounts of pre-conversions reserved for launching ID
  "idregistrationfees" : n,           (number) base cost of IDs for this currency namespace in this currency
  "idreferrallevels" : n,             (int) levels of ID referrals (only for native PBaaS chains and IDs)
  "idimportfees" : n,                 (number) fees required to import an ID to this system (only for native P
BaaS chains and IDs)
  "eras" : "[obj, ...]",              (objarray) different chain phases of rewards and convertibility
  {
    "reward" : "[n, ...]",            (int) reward start for each era in native coin
    "decay" : "[n, ...]",             (int) exponential or linear decay of rewards during each era
    "halving" : "[n, ...]",           (int) blocks between halvings during each era
    "eraend" : "[n, ...]",            (int) block marking the end of each era
    "eraoptions" : "[n, ...]",        (int) options (reserved)
  }
  "nodes"      : "[obj, ..]",    (objectarray, optional) up to 8 nodes that can be used to connect to the blockch
ain      [{
       "nodeidentity" : "txid", (string,  optional) internet, TOR, or other supported address for node
       "paymentaddress" : n,     (int,     optional) rewards payment address
     }, .. ]
  "lastconfirmedcurrencystate" : {
   }
  "besttxid" : "txid"
   }
  "confirmednotarization" : {
   }
  "confirmedtxid" : "txid"
}
```

Examples:

```
> verus getcurrency "chainname"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getcurrency", "params":
["chainname"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getcurrencyconverters ["currency1","currency2",...]`

Retrieves all currencies that have at least 1000 VRSC in reserve, are >10% VRSC reserve ratio, and have all listed currencies as reserves

Arguments:

```
  ["currencyname"    : "string", ...]  (string list, one or more) all selected currencies are returned with their
current state
```

Result:

```
  "[{currency1}, {currency2}]" : "array of objects" (string) All currencies and the last notarization, which are
valid converters.
```

Examples:

```
> verus getcurrencyconverters '["currency1","currency2",...]'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getcurrencyconverters", "p
arams": ['["currency1","currency2",...]'] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getcurrencystate "currencynameorid" ("n") ("connectedsystemid")`

Returns the total amount of preconversions that have been confirmed on the blockchain for the specified chain.

Arguments:

1. "currencynameorid" (string) name or i-address of currency in question
2. "n" or "m,n" or "m,n,o" (int or string, optional) height or inclusive range with optional step at which to get the currency state. If not specified, the latest currency state and height is returned
3. "connectedchainid" (string) optional

Result:

```
[
  {
     "height": n,
     "blocktime": n,
     "currencystate": {
       "flags" : n,
       "initialratio" : n,
       "initialsupply" : n,
       "emitted" : n,
       "supply" : n,
       "reserve" : n,
       "currentratio" : n,
     }
   },
]
```

Examples:

```
> verus getcurrencystate "currencynameorid" ("n") ("connectedchainid")
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getcurrencystate", "param
s": ["currencynameorid" ("n") ("connectedchainid")] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getcurrencytrust '["currencyid",...]'`

Arguments

```
"["currencyid",...]"                              (strarray, optional) if specified, only returns rating
values for specified currencies, otherwise all
```

Result

```
{
  "setratings":{"id":JSONRatingObject,...},        (jsonobj) an ID/ratings key/value object
  "currencytrustmode":<n>                          (int) 0 = no restriction on sync, 1 = only sync to IDs rated
approved, 2 = sync to all IDs but those on block list
}
```

Examples:

```
> verus getcurrencytrust '["currencyid",...]'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getcurrencytrust", "param
s": ['["currencyid",...]'] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getexports "chainname" (heightstart) (heightend)`

Returns pending export transfers to the specified currency from start height to end height if specified

$ Arguments:

```
1. "chainname"                  (string, optional)  name of the chain to look for. no parameter returns curren
t chain in daemon.
2. "heightstart"                (int, optional)     default=0 only return exports at or above this height
3. "heightend"                  (int, optional)     dedfault=maxheight only return exports below or at this he
ight
```

Result:

```
[
  {
    "blockheight": n,                  (numeric) block height
    "exportid": "hash",                (hexstring) transaction ID
    "description": {
      "version": n,                    (int) version of this chain definition
      "exportcurrencyid": "xxxx",      (text) i-address of the currency ID
      "numinputs": n,                  (numeric) Amount of inputs in this export
      "totalamounts": {
        "xxxx": xxx.xxxxxxxx,          (text) i-address of the currency ID, (numeric) the total value of the ex
port
                                       of the currency
        ....
      },
      "totalfees": {
        "xxxx": xxx.xxxxxxxx,          (text) i-address of the currency ID, (numeric) the total fees for the ex
port
                                       of the currency
        ....
      }
    },
    "transfers": [
      {
        "version": n,                  (int) version of this chain definition
        "currencyid": "xxxx",          (text) i-address of the currency ID
        "value": n,                    (numeric) the value this transaction
        "flags": n,                    (hex) flags for the currency
        "preconvert": true|false,      (boolean) Shows whether the transaction is a preconversion
        "fees": xxx.xxxxxxxx,          (numeric) the fees for this transaction
        "destinationcurrencyid": "xxxx", (text) i-address of the currency ID
        "destination": "xxxx"          (text) address of the transaction
      },
      ....
    ]
  },
  {},
  ...
]
```

Examples:

```
> verus getexports "chainname"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getexports", "params": ["c
hainname"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getimports "chainname" (startheight) (endheight)`

Returns all imports into a specific currency, optionally that were imported between a specific block range.

Arguments:

```
1. "chainname"                  (string, optional)  name of the chain to look for. no parameter returns curren
t chain in daemon.
2. "heightstart"                (int, optional)     default=0 only return exports at or above this height
3. "heightend"                  (int, optional)     default=maxheight only return exports below or at this hei
ght
```

Result:

```
[
  {
    "blockheight": n,                  (numeric) block height
    "importid": "hash",                (hexstring) transaction ID
    "description": {
      "version":n,                     (int) version of this chain definition
      "sourcesystemid": "xxxx",        (text) i-address of the source system
      "importcurrencyid": "xxxx",      (text) i-address of the currency ID
      "valuein": {
        "xxxx": xxx.xxxxxxxx,          (text) i-address of the currency ID, (numeric) the total value of the im
port
                                       of the currency
        ....
      },
      "tokensout": {
        "xxxx": xxx.xxxxxxxx,          (text) i-address of the currency ID, (numeric) the total value of the im
port
                                       into the currency into this currency
      }
    },
    "transfers": [
      {
        "version": n,                  (int) version of this chain definition
        "currencyid": "xxxx",          (text) i-address of the currency ID
        "value": n,                    (numeric) the value this transaction
        "flags": n,                    (hex) flags for the currency
        "preconvert": true|false,      (boolean) Shows whether the transaction is a preconversion
        "fees": xxx.xxxxxxxx,          (numeric) the fees for this transaction
        "destinationcurrencyid": "xxxx", (text) i-address of the currency ID
        "destination": "xxxx"          (text) address of the transaction
      },
      ....
      }
    ]
  },
  {},
  ...
]
```

Examples:

```
> verus getimports "chainname"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getimports", "params": ["c
hainname"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getinitialcurrencystate "name"`

Returns the total amount of preconversions that have been confirmed on the blockchain for the specified PBaaS chain.
This should be used to get information about chains that are not this chain, but are being launched by it.

Arguments:

```
   "name"                  (string, required) name or chain ID of the chain to get the export transactions for
```

Result:

```
   [
     {
       "flags": n,                     (hex) flags for the currency
       "currencyid": "xxxx",           (text) i-address of the currency ID
       "reservecurrencies": [
         {
                "currencyid": "xxxx",        (text) i-address of the currency ID
                "weight": n,                 (numeric) The weight of this currence
                "reserves": xxx.xxxxxxxx,    (numeric) The initial amount of this currency in reserve
                "priceinreserve": n          (numeric) The initial conversion rate of this reserve currency
            },
       ],
       "initialsupply" : xxx.xxxxxxxx,  (numeric) The initial supply of this currency
       "emitted" : xxx.xxxxxxx,         (numeric) The currently emitted coins of this currency
       "supply" : xxx.xxxxxxxx,         (numeric) The current supply of this currency
       "currencies": {
         "xxxx": {                     (text) i-address of the currency ID
            "reservein": xxx.xxxxxxxx,
            "nativein": 0.00000000,
            "reserveout": 249.99900299,
            "lastconversionprice": 1.00000000,
            "viaconversionprice": 0.99981249,
            "fees": 250.06291562,
            "conversionfees": 250.06251562
            },
       },
     },
   ]
```

Examples:

```
> verus getinitialcurrencystate name
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getinitialcurrencystate",
"params": [name] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getlastimportfrom "systemname"`

Returns the last import from a specific originating system.

Arguments:

```
   "systemname"              (string, required) name or ID of the system to retrieve the last import from
```

Result:

```
{
   "lastimport" :                  (object) last import from the indicated system on this chain
     {
     }
   "lastconfirmednotarization" :   (object) last confirmed notarization of the indicated system on this chain
     {
     }
}
```

Examples:

```
> verus getlastimportfrom "systemname"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getlastimportfrom", "param
s": "systemname" }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getlaunchinfo "currencyid"`

Returns the launch notarization data and partial transaction proof of the launch notarization for the specifed currencyid.

Arguments:

1. "currencyid" (string, required) the hex-encoded ID or string name search for notarizations on

Result:

```
{
  "currencydefinition" : {},     (json) Full currency definition
  "txid" : "hexstr",             (hexstr) transaction ID
  "voutnum" : "n",               (number) vout index of the launch notarization
  "transactionproof" : {},       (json) Partial transaction proof of the launch transaction and output
  "launchnotarization" : {},     (json) Final CPBaaSNotarization clearing launch or refund
  "notarynotarization" : {},     (json) Current notarization of this chain
}
```

Examples:

```
> verus getlaunchinfo "currencyid"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getlaunchinfo", "params":
["currencyid"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getnotarizationdata "currencyid" (getevidence) (separatecounterevidence)`

Returns the latest PBaaS notarization data for the specifed currencyid.

Arguments

```
1. "currencyid"                (string, required) the hex-encoded ID or string name search for notarizations on
2. "(getevidence)"               (bool, optional)    if true, returns notarization evidence as well as other da
ta
3. "(separatecounterevidence)"   (bool, optional)    if true, counter-evidence is processed and returned with pro
ofroots
```

Result:

```
{
  "version" : n,               (numeric) The notarization protocol version
}
```

Examples:

```
> verus getnotarizationdata "currencyid"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getnotarizationdata", "par
ams": ["currencyid"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

'getnotarizationproofs {json object}'

Returns proofs to a caller for requested challenges. Some proofs can either independently or in combination with other proofs over time invalidate or force a competing chain to provide more proofs in order to confirm any pending cross-chain notarization of an alternate chain that may not agree with us.

- It is not valid to have a challenge request with both confirmnotarization and confirmroot.

Arguments:

"challengerequests" (array, required) one or more challenge requests for unconfirmed notarizations on a bridged system

```
[
  {
    "type":"vrsc::evidence.skipchallenge" || "iCwxpRL6h3YeCRtGjgQSsqoKdZCuM4Dxaf",
    "evidence":{CNotaryEvidence},
    "entropyhash":"hex",
    "proveheight":n,
    "atheight":n
  },
  {
    "type":"vrsc::evidence.validitychallenge" || "iCPb8ywQna7jYV2SHrGZ6vQMj7kuyWFxvb",
    "evidence":{CNotaryEvidence},
    "entropyhash":"hex",
    "fromheight":n,
    "toheight":n,
    "challengedroot":{CPRoofRoot},
    "confirmnotarization":{expectednotarization}, | "confirmroot":{CPRoofRoot}
  },
  {"type":"vrsc::evidence.primaryproof" || "iKDesmiEkEjDG61nQSZJSGhWvC8x8xA578",
     "priornotarizationref":{CUTXORef} || "priorroot":{CProofRoot
  },
  "challengeroots":
  [
    {
      "indexkey":{object},
      "proofroot":{CProofRoot}
    },
  ...
  ],
  "evidence":{CNotaryEvidence},
  "entropyhash":"hex",
  "confirmnotarization":{newnotarization}, |
  "confirmroot":{CPRoofRoot}},
  "fromheight":n,
  "toheight":n},
  ...
]
```

Results:

```
{"evidence":[{CNotaryEvidence}, ...]   (array) notary evidence challenges, including proofs for challenges reques
ted
```

Examples:

```
> verus getnotarizationproofs '[{"type":"iCwxpRL6h3YeCRtGjgQSsqoKdZCuM4Dxaf", "evidence":{CNotaryEvidence}, "prov
eheight":n, "atheight":n}, ...]'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getnotarizationproofs", "p
arams": [[{"type":"iCwxpRL6h3YeCRtGjgQSsqoKdZCuM4Dxaf", "evidence":{CNotaryEvidence}, "proveheight":n, "atheigh
t":n}, ...]] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getpendingtransfers "chainname"`

Returns all pending transfers for a particular chain that have not yet been aggregated into an export.

Arguments:

```
1. "chainname"                (string, optional) name of the chain to look for. no parameter returns current
chain in daemon.
```

Result:

```
  {
  }
```

Examples:

```
> verus getpendingtransfers "chainname"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getpendingtransfers", "par
ams": ["chainname"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getreservedeposits "currencyname"`

Returns all deposits under control of the specified currency or chain. If the currency is of an external system or chain, all deposits will be under the control of that system or chain only, not its independent currencies.

Arguments:

1. "currencyname" (string, optional) full name or i-ID of controlling currency

Result:

```
  {
  }
```

Examples:

```
> verus getreservedeposits "currencyname"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getreservedeposits", "params": ["currencyname"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

getsaplingtree "n"

Returns the entries for a light wallet Sapling tree state.

Arguments:

```
   "n" or "m,n" or "m,n,o"         (int or string, optional) height or inclusive range with optional step at which to get the Sapling tree state. If not specified, the latest currency state and height is returned
```

Result:

```
   [
       {
           "network": "VRSC",
           "height": n,
           "hash": "hex"
           "time": n,
           "tree": "hex"
       },
   ]
```

Examples:

```
> verus getsaplingtree name
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getsaplingtree", "params": [name] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

getsaplingtree "n" ***NEW***

Returns the entries for a light wallet Sapling tree state.

Arguments:

1. "n" or "m,n" or "m,n,o" (int or string, optional) height or inclusive range with optional step at which to get the Sapling tree state

   ```
                                                 If not specified, the latest currency state and height
   is returned
   ```

   ### Result:

   ```
   [
     {
         "network": "VRSC",
         "height": n,
         "hash": "hex"
         "time": n,
         "tree": "hex"
     },
   ]
   ```

   ### Examples:

   ```
   > verus getsaplingtree name
   > curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getsaplingtree", "params": [name] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
   ```

listcurrencies ({query object}) startblock endblock

Returns a complete definition for any given chain if it is registered on the blockchain. If the chain requested is NULL, chain definition of the current chain is returned.

Arguments:

```
{                                   (json, optional) specify valid query conditions
   "launchstate" : ("prelaunch" | "launched" | "refund" | "complete") (optional) return only currencies in that state
   "systemtype" : ("local" | "gateway" | "pbaas")
   "converter": bool                (bool, optional) default false, only return fractional currency converters
}
```

Result:

```
[
  {
    "version" : n,                        (int) version of this chain definition
    "name" : "string",                    (string) name or symbol of the chain, same as passed
    "fullyqualifiedname" : "string",      (string) name or symbol of the chain with all parent namespaces, separated by "."
    "currencyid" : "i-address",           (string) string that represents the currency ID, same as the ID behind the currency
    "currencyidhex" : "hex",              (string) hex representation of currency ID, getcurrency API supports "hex:currencyidhex"
    "parent" : "i-address",               (string) parent blockchain ID
    "systemid" : "i-address",             (string) system on which this currency is considered to run
    "launchsystemid" : "i-address",       (string) system from which this currency was launched
    "notarizationprotocol" : n            (int) protocol number that determines variations in cross-chain or bridged notarizations
    "proofprotocol" : n                   (int) protocol number that determines variations in cross-chain or bridged proofs
    "startblock" : n,                     (int) block # on this chain, which must be notarized into block one of the chain
    "endblock" : n,                       (int) block # after which, this chain's useful life is considered to be over
    "currencies" : "["i-address", ...]",  (stringarray) currencies that can be converted to this currency at launch or makeup a liquidity basket
    "weights" : "[n, ...]",               (numberarray) relative currency weights (only returned for a liquidity basket)
    "conversions" : "[n, ...]",           (numberarray) pre-launch conversion rates for non-fractional currencies
    "minpreconversion" : "[n, ...]",      (numberarray) minimum amounts required in pre-conversions for currency to launch
    "currencies" : "["i-address", ...]",  (stringarray) currencies that can be converted to this currency at launch or makeup a liquidity basket
    "currencynames" : "{"i-address":"fullname",...}", (obj) i-addresses mapped to fully qualified names of all sub-currencies
    "initialsupply" : n,                  (number) initial currency supply for fractional currencies before preallocation or issuance
    "prelaunchcarveout" : n,              (number) pre-launch percentage of proceeds for fractional currency sent to launching ID
    "preallocations" : "[{"i-address":n}, ...]", (objarray) VerusIDs and amounts for pre-allocation at launch
    "initialcontributions" : "[n, ...]",  (numberarray) amounts of pre-conversions reserved for launching ID
    "idregistrationfees" : n,             (number) base cost of IDs for this currency namespace in this currency
    "idreferrallevels" : n,               (int) levels of ID referrals (only for native PBaaS chains and IDs)
    "idimportfees" : n,                   (number) fees required to import an ID to this system (only for native PBaaS chains and IDs)
    "eras" : "[obj, ...]",                (objarray) different chain phases of rewards and convertibility
    {
      "reward" : "[n, ...]",              (int) reward start for each era in native coin
      "decay" : "[n, ...]",               (int) exponential or linear decay of rewards during each era
      "halving" : "[n, ...]",             (int) blocks between halvings during each era
      "eraend" : "[n, ...]",              (int) block marking the end of each era
      "eraoptions" : "[n, ...]",          (int) options (reserved)
    }
    "nodes"      : "[obj, ..]",           (objectarray, optional) up to 8 nodes that can be used to connect to the blockchain      [{
         "nodeidentity" : "txid",         (string,  optional) internet, TOR, or other supported address for node
         "paymentaddress" : n,            (int,     optional) rewards payment address
       }, .. ]
    "lastconfirmedcurrencystate" : {
     }
    "besttxid" : "txid"
     }
    "confirmednotarization" : {
     }
    "confirmedtxid" : "txid"
  }, ...
]
```

Examples:

```
> verus listcurrencies true
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "listcurrencies", "params": [true] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

refundfailedlaunch "currencyid"

Refunds any funds sent to the chain if they are eligible for refund.
This attempts to refund all transactions for all contributors.

Arguments:

```
"currencyid"           (hex or chain name, required) the chain to refund contributions to
```

Result:

Examples:

```
> verus refundfailedlaunch "currencyid"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "refundfailedlaunch", "params": ["currencyid"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

sendcurrency "fromaddress" '[{"address":... ,"amount":...},...]' (minconfs) (feeamount) (returntxtemplate)

This sends one or many Verus outputs to one or many addresses on the same or another chain.
Funds are sourced automatically from the current wallet, which must be present, as in sendtoaddress.
If "fromaddress" is specified, all funds will be taken from that address, otherwise funds may come from any source set of UTXOs controlled by the wallet.

Arguments:

1. "fromaddress" (string, required) The VerusID or address to send the funds from. "*", "R*" and "i*" wildcards are supported. Also `"idname@:private"` can be used to source from a balance of a private address attached to an ID.
2. "outputs" (array, required) An array of json objects representing currencies, amounts, and destinations to send.

   ```
    [{
      "currency": "name"     (string,  required) Name of the source currency to send in this output, defaults to native of chain
      "amount":amount        (numeric, required) The numeric amount of currency, denominated in source currency
      "convertto":"name",    (string,  optional) Valid currency to convert to, either a reserve of a fractional, or fractional
      "exportto":"name",     (string,  optional) Valid chain or system name or ID to export to
      "exportid":"false",    (bool,    optional) if cross-chain ID, export the ID to the destination chain (will cost to export)
      "exportcurrency":"false", (bool, optional) if cross-chain export, export the currency definition (will cost to export)
      "feecurrency":"name",  (string,  optional) Valid currency that should be pulled from the current wallet and used to pay fee
      "via":"name",          (string,  optional) If source and destination currency are reserves, via is a common fractional to convert through
      "address":"dest"       (string,  required) The address and optionally chain/system after the "@" as a system specific destination
      "refundto":"dest"      (string,  optional) For pre-conversions, this is where refunds will go, defaults to fromaddress
      "memo":memo            (string,  optional) If destination is a zaddr (not supported on testnet), a string message (not hexadecimal) to include.
      "preconvert":"false",  (bool,    optional) convert to currency at market price (default=false), only works if transaction is mined before start of currency
      "burn":"false",        (bool,    optional) destroy the currency and subtract it from the supply. Currency must be a token.
      "mintnew":"false",     (bool,    optional) if the transaction is sent from the currency ID of a centralized currency, this creates new currency to send
    }, ... ]
   ```

3. "minconf" (numeric, optional, default=1) only use funds confirmed at least this many times.
4. "feeamount" (numeric, optional) specific fee amount requested instead of default miner's fee.
5. "returntxtemplate (boolean, optional, default false) if true, returns transaction info without submitting the transaction to the chain.

Result:

1. "operationid" (string) An operationid to pass to z_getoperationstatus to get the result of the operation.
2. If (returntxtemplate) is true

   ```
   {
   "outputtotals" : {currencyvaluemap}   Total outputs in all currencies the need to be input to the transaction
   "hextx" : "hexstring"                 The transaction with all specified outputs and no inputs
   }
   ```

Examples:

```
> verus sendcurrency "*" '[{"currency":"btc","address":"RRehdmUV7oEAqoZnzEGBH34XysnWaBatct" ,"amount":500.0},...]'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "sendcurrency", "params": ["bob@", [{"currency":"btc", "address":"alice@quad", "amount":500.0},...]] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

setcurrencytrust '{"clearall": bool, "setratings":[{"currencyid":JSONRatingObject},...], "removeratings":["currencyid",...], "currencytrustmode":<n>}'

Arguments

```
{
  "clearall": bool                                (bool, optional) clears all wallet currency trust lists before adding, removing, or trust mode operations
  "setratings":[{"currencyid":JSONRatingObject},...] (objarray, optional) replaces ratings for specified currencies with those given
  "removeratings":["currencyid",...]              (strarray, optional) erases ratings for currencies specified
  "currencytrustmode": <n>                        (number, optional) 0 = spend/list all currencies, 1 = only spend/list those rated approved, 2 = spend/list all but those on block list
}
```

Result

no return on success, else error

submitacceptednotarization "{earnednotarization}" "{notaryevidence}"

Finishes an almost complete notarization transaction based on the notary chain and the current wallet or pubkey.

If successful in submitting the transaction based on all rules, a transaction ID is returned, otherwise, NULL.

Arguments:

"earnednotarization" (object, required) notarization earned on the other system, which is the basis for this
"notaryevidence" (object, required) evidence and notary signatures validating the notarization

Result:

txid (hexstring) transaction ID of submitted transaction

Examples:

```
> verus submitacceptednotarization "{earnednotarization}" "{notaryevidence}"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "submitacceptednotarization", "params": ["{earnednotarization}" "{notaryevidence}"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

submitchallenges [json array]

Submits one or more cryptographic challenges to existing, unconfirmed notarizations, proving the existence of an alternate chain. Whether the alternate chain has more power than the chain with a pending notarization is not required, only that it has moved forward multiple blocks since the prior notarization.

Requires that the local wallet have funds for fees.

Arguments:

"challenges" (array, required) one or more challenges to unconfirmed notarizations on this system

```
[
  {
    "type":"vrsc::evidence.skipchallenge" || "iCwxpRL6h3YeCRtGjgQSsqoKdZCuM4Dxaf" ||
                        "type":"vrsc::evidence.validitychallenge" || "iCPb8ywQna7jYV2SHrGZ6vQMj7kuyWFxvb",
    "notarizationref":{"txid":"hexvalue","voutnum":n},
    "forkroot":{},
    "challengeroot":{},
    "evidence":{}
  },
  {...},
   ...
]
```

Result:

```
[{"txid":"hex"}, {"error":"errorstring"}, ...]   (array) results of submitted challenge transactions
```

Examples:

```
> verus submitchallenges '[{"notarizationref":{"txid":"hexvalue","voutnum":n},"challengeroot":{},"evidence":{}}]'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "submitchallenges", "params": [[{"notarizationref":{"txid":"hexvalue","voutnum":n},"challengeroot":{},"evidence":{}}]] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

submitimports '{"sourcesystemid":"systemid", "notarizationtxid":"txid", "notarizationtxoutnum":n, "exports":[{"txid":"hexid", "txoutnum":n, "partialtransactionproof":"hexstr", Fresendw"transfers": [{transfer1}, {transfer2},...]}, ...]}'

Accepts a set of exports from another system to post to the VRSC network.

Arguments:

```
  {
    "sourcesystemid":"systemid"       ()
    "notarizationtxid":"txid"         ()
    "notarizationtxoutnum":n            ()
    "exports": [{
       "height": n,
       "txid": "hexid",
       "txoutnum": n,
       "partialtransactionproof": "hexstr",
       "transfers": [{transfer1}, {transfer2},...]
    }, ...]
  }
```

Result:

```
  [{
     "currency": "currencyid"
     "txid": "hexid",
     "txoutnum": n
  }, ...]
```

Examples:

```
> verus submitimports {"sourcesystemid":"systemid", "notarizationtxid":"txid", "notarizationtxoutnum":n, "exports":[{"height":n, "txid":"hexid", "txoutnum":n, "partialtransactionproof":"hexstr", "transfers": [{transfer1}, {transfer2},...]}, ...]}
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "submitimports", "params": [{"sourcesystemid":"systemid", "notarizationtxid":"txid", "notarizationtxoutnum":n, "exports":[{"height":n, "txid":"hexid", "txoutnum":n, "partialtransactionproof":"hexstr", "transfers": [{transfer1}, {transfer2},...]}, ...]}] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

submitmergedblock "hexdata" ("jsonparametersobject")

Attempts to submit one more more new blocks to one or more networks.
Each merged block submission may be valid for Verus and/or PBaaS merge mined chains.
The submitted block consists of a valid block for this chain, along with embedded headers of other PBaaS merge mined chains.
If the hash for this header meets targets of other chains that have been added with 'addmergedblock', this API will
submit those blocks to the specified URL endpoints with an RPC 'submitmergedblock' request.
Attempts to submit one more more new blocks to one or more networks.

Arguments

1. "hexdata" (string, required) the hex-encoded block data to submit

Results:

```
   { rejected: "reject reason" }
```

Submission to our chain and PBaaS chains

```
{ blockhash: "hex", accepted: true, pbaas_submissions: { "Quantum":"chainID_hex", ... } }
```

Submission to only PBaaS chains

```
{ blockhash: "hex", accepted: "pbaas", pbaas_submissions: { "Quantum":"chainID_hex", ... } }
```

Examples:

```
> verus submitmergedblock "mydata"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "submitmergedblock", "params": ["mydata"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

Network

`addnode "node" "add|remove|onetry"`

Attempts add or remove a node from the addnode list.
Or try a connection to a node once.

Arguments:

1. "node" (string, required) The node (see getpeerinfo for nodes)
2. "command" (string, required) 'add' to add a node to the list, 'remove' to remove a node from the list, 'onetry' to try a connection to the node once

Examples:

```
> verus addnode "192.168.0.6:8233" "onetry"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "addnode", "params": ["192.168.0.6:8233", "onetry"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`clearbanned`

Clear all banned IPs.

Examples:

```
> verus clearbanned
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "clearbanned", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`disconnectnode "node"`

Immediately disconnects from the specified node.

Arguments:

1. "node" (string, required) The node (see getpeerinfo for nodes)

Examples:

```
> verus disconnectnode "192.168.0.6:8233"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "disconnectnode", "params": ["192.168.0.6:8233"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getaddednodeinfo dns ( "node" )`

Returns information about the given added node, or all added nodes (note that onetry addnodes are not listed here)
If dns is false, only a list of added nodes will be provided, otherwise connected information will also be available.

Arguments:

1. dns (boolean, required) If false, only a list of added nodes will be provided, otherwise connected information will also be available.
2. "node" (string, optional) If provided, return information about this specific node, otherwise all nodes are returned.

Result:

```
[
  {
    "addednode" : "192.168.0.201",   (string) The node ip address
    "connected" : true|false,          (boolean) If connected
    "addresses" : [
       {
         "address" : "192.168.0.201:27485",  (string) The Verus server host and port
         "connected" : "outbound"           (string) connection, inbound or outbound
       }
       ,...
     ]
  }
  ,...
]
```

Examples:

```
> verus getaddednodeinfo true
> verus getaddednodeinfo true "192.168.0.201"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getaddednodeinfo", "params": [true, "192.168.0.201"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getconnectioncount`

Returns the number of connections to other nodes.

Result:

n (numeric) The connection count

Examples:

```
> verus getconnectioncount
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getconnectioncount", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getdeprecationinfo`

Returns an object containing current version and deprecation block height. Applicable only on mainnet.

Result:

```
{
  "version": xxxxx,                      (numeric) the server version
  "subversion": "/MagicBean:x.y.z[-v]/",     (string) the server subversion string
  "deprecationheight": xxxxx,            (numeric) the block height at which this version will deprecate and shut down
}
```

Examples:

```
> verus getdeprecationinfo
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getdeprecationinfo", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getnettotals`

Returns information about network traffic, including bytes in, bytes out,
and current time.

Result:

```
{
  "totalbytesrecv": n,   (numeric) Total bytes received
  "totalbytessent": n,   (numeric) Total bytes sent
  "timemillis": t        (numeric) Total cpu time
}
```

Examples:

```
> verus getnettotals
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getnettotals", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getnetworkinfo`

Returns an object containing various state info regarding P2P networking.

Result:

```
{
  "version": xxxxx,                      (numeric) the server version
  "subversion": "/MagicBean:x.y.z[-v]/",     (string) the server subversion string
  "protocolversion": xxxxx,              (numeric) the protocol version
  "localservices": "xxxxxxxxxxxxxxxx", (string) the services we offer to the network
  "timeoffset": xxxxx,                   (numeric) the time offset
  "connections": xxxxx,                  (numeric) the number of connections
  "networks": [                          (array) information per network
  {
    "name": "xxx",                     (string) network (ipv4, ipv6 or onion)
    "limited": true|false,               (boolean) is the network limited using -onlynet?
    "reachable": true|false,             (boolean) is the network reachable?
    "proxy": "host:port"               (string) the proxy that is used for this network, or empty if none
  }
  ,...
  ],
  "relayfee": x.xxxxxxxx,                (numeric) minimum relay fee for non-free transactions in VRSC/kB
  "localaddresses": [                    (array) list of local addresses
  {
    "address": "xxxx",                 (string) network address
    "port": xxx,                         (numeric) network port
    "score": xxx                         (numeric) relative score
  }
  ,...
  ]
  "warnings": "..."                    (string) any network warnings (such as alert messages)
}
```

Examples:

```
> verus getnetworkinfo
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getnetworkinfo", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getpeerinfo`

Returns data about each connected network node as a json array of objects.

Result:

```
[
  {
    "id": n,                   (numeric) Peer index
    "addr":"host:port",      (string) The ip address and port of the peer
    "addrlocal":"ip:port",   (string) local address
    "services":"xxxxxxxxxxxxxxxx",   (string) The services offered
    "lastsend": ttt,           (numeric) The time in seconds since epoch (Jan 1 1970 GMT) of the last send
    "lastrecv": ttt,           (numeric) The time in seconds since epoch (Jan 1 1970 GMT) of the last receive
    "bytessent": n,            (numeric) The total bytes sent
    "bytesrecv": n,            (numeric) The total bytes received
    "conntime": ttt,           (numeric) The connection time in seconds since epoch (Jan 1 1970 GMT)
    "timeoffset": ttt,         (numeric) The time offset in seconds
    "pingtime": n,             (numeric) ping time
    "pingwait": n,             (numeric) ping wait
    "version": v,              (numeric) The peer version, such as 170002
    "subver": "/MagicBean:x.y.z[-v]/",  (string) The string version
    "inbound": true|false,     (boolean) Inbound (true) or Outbound (false)
    "startingheight": n,       (numeric) The starting height (block) of the peer
    "banscore": n,             (numeric) The ban score
    "synced_headers": n,       (numeric) The last header we have in common with this peer
    "synced_blocks": n,        (numeric) The last block we have in common with this peer
    "inflight": [
       n,                        (numeric) The heights of blocks we're currently asking from this peer
       ...
    ]
  }
  ,...
]
```

Examples:

```
> verus getpeerinfo
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getpeerinfo", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`listbanned`

List all banned IPs/Subnets.

Examples:

```
> verus listbanned
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "listbanned", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`ping`

Requests that a ping be sent to all other nodes, to measure ping time.
Results provided in getpeerinfo, pingtime and pingwait fields are decimal seconds.
Ping command is handled in queue with all other commands, so it measures processing backlog, not just network ping.

Examples:

```
> verus ping
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "ping", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`setban "ip(/netmask)" "add|remove" (bantime) (absolute)`

Attempts add or remove a IP/Subnet from the banned list.

Arguments:

1. "ip(/netmask)" (string, required) The IP/Subnet (see getpeerinfo for nodes ip) with a optional netmask (default is /32 = single ip)
2. "command" (string, required) 'add' to add a IP/Subnet to the list, 'remove' to remove a IP/Subnet from the list
3. "bantime" (numeric, optional) time in seconds how long (or until when if [absolute] is set) the ip is banned (0 or empty means using the default time of 24h which can also be overwritten by the -bantime startup argument)
4. "absolute" (boolean, optional) If set, the bantime must be a absolute timestamp in seconds since epoch (Jan 1 1970 GMT)

Examples:

```
> verus setban "192.168.0.6" "add" 86400
> verus setban "192.168.0.0/24" "add"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "setban", "params": ["192.168.0.6", "add" 86400] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

Rawtransactions

`createrawtransaction [{"txid":"id","vout":n},...] {"address":amount,...} ( locktime ) ( expiryheight )`

Create a transaction spending the given inputs and sending to the given addresses.
Returns hex-encoded raw transaction.
Note that the transaction's inputs are not signed, and
it is not stored in the wallet or transmitted to the network.

Arguments:

1. "transactions" (string, required) A json array of json objects

```
[
  {
    "txid":"id",    (string, required) The transaction id
    "vout":n        (numeric, required) The output number
    "sequence":n    (numeric, optional) The sequence number
  }
  ,...
]
```

2. "addresses" (string, required) a json object with addresses as keys and amounts as values

```
{
  "address": x.xxx   (numeric, required) The key is the Komodo address, the value is the VRSC amount
  ,...
}
```

3. locktime (numeric, optional, default=0) Raw locktime. Non-0 value also locktime-activates inputs
4. expiryheight (numeric, optional, default=nextblockheight+20 (pre-Blossom) or nextblockheight+40 (post-Blossom)) Expiry height of transaction (if Overwinter is active)

Result:

"transaction" (string) hex string of the transaction

Examples

```
> verus createrawtransaction "[{\"txid\":\"myid\",\"vout\":0}]" "{\"address\":0.01}"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "createrawtransaction", "params": ["[{\"txid\":\"myid\",\"vout\":0}]", "{\"address\":0.01}"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`decoderawtransaction "hexstring"`

Return a JSON object representing the serialized, hex-encoded transaction.

Arguments:

1. "hex" (string, required) The transaction hex string

Result:

```
{
  "txid" : "id",        (string) The transaction id
  "overwintered" : bool   (boolean) The Overwintered flag
  "version" : n,          (numeric) The version
  "versiongroupid": "hex"   (string, optional) The version group id (Overwintered txs)
  "locktime" : ttt,       (numeric) The lock time
  "expiryheight" : n,     (numeric, optional) Last valid block height for mining transaction (Overwintered txs)
  "vin" : [               (array of json objects)
     {
       "txid": "id",    (string) The transaction id
       "vout": n,         (numeric) The output number
       "scriptSig": {     (json object) The script
         "asm": "asm",  (string) asm
         "hex": "hex"   (string) hex
       },
       "sequence": n     (numeric) The script sequence number
     }
     ,...
  ],
  "vout" : [             (array of json objects)
     {
       "value" : x.xxx,            (numeric) The value in VRSC
       "n" : n,                    (numeric) index
       "scriptPubKey" : {          (json object)
         "asm" : "asm",          (string) the asm
         "hex" : "hex",          (string) the hex
         "reqSigs" : n,            (numeric) The required sigs
         "type" : "pubkeyhash",  (string) The type, eg 'pubkeyhash'
         "addresses" : [           (json array of string)
           "RTZMZHDFSTFQst8XmX2dR4DaH87cEUs3gC"   (string) komodo address
           ,...
         ]
       }
     }
     ,...
  ],
  "vjoinsplit" : [        (array of json objects, only for version >= 2)
     {
       "vpub_old" : x.xxx,         (numeric) public input value in KMD
       "vpub_new" : x.xxx,         (numeric) public output value in KMD
       "anchor" : "hex",         (string) the anchor
       "nullifiers" : [            (json array of string)
```

```
        "hex"                    (string) input note nullifier
        ,...
      ],
      "commitments" : [          (json array of string)
        "hex"                    (string) output note commitment
        ,...
      ],
      "onetimePubKey" : "hex",  (string) the onetime public key used to encrypt the ciphertexts
      "randomSeed" : "hex",     (string) the random seed
      "macs" : [                 (json array of string)
        "hex"                    (string) input note MAC
        ,...
      ],
      "proof" : "hex",          (string) the zero-knowledge proof
      "ciphertexts" : [          (json array of string)
        "hex"                    (string) output note ciphertext
        ,...
      ]
    }
    ,...
  ],
}
```

Examples:

```
> verus decoderawtransaction "hexstring"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "decoderawtransaction", "params": ["hexstring"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`decodescript "hex"`

Decode a hex-encoded script.

Arguments:

1. "hex" (string) the hex encoded script

Result:

```
{
  "asm":"asm",   (string) Script public key
  "hex":"hex",   (string) hex encoded public key
  "type":"type", (string) The output type
  "reqSigs": n,    (numeric) The required signatures
  "addresses": [   (json array of string)
     "address"     (string) Komodo address
     ,...
  ],
  "p2sh","address" (string) script address
}
```

Examples:

```
> verus decodescript "hexstring"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "decodescript", "params": ["hexstring"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`fundrawtransaction "hexstring"`

Add inputs to a transaction until it has enough in value to meet its out value.
This will not modify existing inputs, and will add one change output to the outputs.
Note that inputs which were signed may need to be resigned after completion since in/outputs have been added.
The inputs added will not be signed, use signrawtransaction for that.

Arguments:

1. "hexstring" (string, required) The hex string of the raw transaction

Result:

```
{
  "hex":       "value", (string)  The resulting raw transaction (hex-encoded string)
  "fee":       n,         (numeric) The fee added to the transaction
  "changepos": n          (numeric) The position of the added change output, or -1
}
"hex"
```

Examples:

Create a transaction with no inputs

```
> verus createrawtransaction "[]" "{\"myaddress\":0.01}"
```

Add sufficient unsigned inputs to meet the output value

```
> verus fundrawtransaction "rawtransactionhex"
```

Sign the transaction

```
> verus signrawtransaction "fundedtransactionhex"
```

Send the transaction

```
> verus sendrawtransaction "signedtransactionhex"
```

`getrawtransaction "txid" ( verbose )`

NOTE: By default this function only works sometimes. This is when the tx is in the mempool
or there is an unspent output in the utxo for this transaction. To make it always work,
you need to maintain a transaction index, using the -txindex command line option.

Return the raw transaction data.

If verbose=0, returns a string that is serialized, hex-encoded data for 'txid'.
If verbose is non-zero, returns an Object with information about 'txid'.

Arguments:

1. "txid" (string, required) The transaction id
2. verbose (numeric, optional, default=0) If 0, return a string, other return a json object

Result (if verbose is not set or set to 0):
"data" (string) The serialized, hex-encoded data for 'txid'

Result (if verbose > 0):

```
{
  "hex" : "data",       (string) The serialized, hex-encoded data for 'txid'
  "txid" : "id",        (string) The transaction id (same as provided)
  "version" : n,          (numeric) The version
  "locktime" : ttt,       (numeric) The lock time
  "expiryheight" : ttt,   (numeric, optional) The block height after which the transaction expires
  "vin" : [               (array of json objects)
     {
       "txid": "id",    (string) The transaction id
       "vout": n,         (numeric)
       "scriptSig": {     (json object) The script
         "asm": "asm",  (string) asm
         "hex": "hex"   (string) hex
       },
       "sequence": n      (numeric) The script sequence number
     }
     ,...
  ],
  "vout" : [              (array of json objects)
     {
       "value" : x.xxx,            (numeric) The value in VRSC
       "n" : n,                    (numeric) index
       "scriptPubKey" : {          (json object)
         "asm" : "asm",          (string) the asm
         "hex" : "hex",          (string) the hex
         "reqSigs" : n,            (numeric) The required sigs
         "type" : "pubkeyhash",  (string) The type, eg 'pubkeyhash'
         "addresses" : [           (json array of string)
           "komodoaddress"          (string) Komodo address
           ,...
         ]
       }
     }
     ,...
  ],
  "vjoinsplit" : [        (array of json objects, only for version >= 2)
     {
       "vpub_old" : x.xxx,         (numeric) public input value in KMD
       "vpub_new" : x.xxx,         (numeric) public output value in KMD
       "anchor" : "hex",         (string) the anchor
       "nullifiers" : [            (json array of string)
         "hex"                     (string) input note nullifier
         ,...
       ],
       "commitments" : [           (json array of string)
         "hex"                     (string) output note commitment
         ,...
       ],
       "onetimePubKey" : "hex",  (string) the onetime public key used to encrypt the ciphertexts
       "randomSeed" : "hex",     (string) the random seed
       "macs" : [                  (json array of string)
         "hex"                     (string) input note MAC
         ,...
       ],
       "proof" : "hex",          (string) the zero-knowledge proof
       "ciphertexts" : [           (json array of string)
         "hex"                     (string) output note ciphertext
         ,...
       ]
     }
     ,...
  ],
  "blockhash" : "hash",   (string) the block hash
  "confirmations" : n,      (numeric) The confirmations
  "time" : ttt,             (numeric) The transaction time in seconds since epoch (Jan 1 1970 GMT)
  "blocktime" : ttt         (numeric) The block time in seconds since epoch (Jan 1 1970 GMT)
}
```

Examples:

```
> verus getrawtransaction "mytxid"
> verus getrawtransaction "mytxid" 1
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getrawtransaction", "params": ["mytxid", 1] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`sendrawtransaction "hexstring" ( allowhighfees )`

Submits raw transaction (serialized, hex-encoded) to local node and network.

Also see createrawtransaction and signrawtransaction calls.

Arguments:

1. "hexstring" (string, required) The hex string of the raw transaction)
2. allowhighfees (boolean, optional, default=false) Allow high fees

Result:

"hex" (string) The transaction hash in hex

Examples:

Create a transaction

```
> verus createrawtransaction "[{\"txid\" : \"mytxid\",\"vout\":0}]" "{\"myaddress\":0.01}"
```

Sign the transaction, and get back the hex

```
> verus signrawtransaction "myhex"
```

Send the transaction (signed hex)

```
> verus sendrawtransaction "signedhex"
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "sendrawtransaction", "params": ["signedhex"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`signrawtransaction "hexstring" ( [{"txid":"id","vout":n,"scriptPubKey":"hex","redeemScript":"hex"},...] ["privatekey1",...] sighashtype )`

Sign inputs for raw transaction (serialized, hex-encoded).
The second optional argument (may be null) is an array of previous transaction outputs that this transaction depends on but may not yet be in the block chain.
The third optional argument (may be null) is an array of base58-encoded private keys that, if given, will be the only keys used to sign the transaction.

Arguments:

1. "hexstring" (string, required) The transaction hex string
2. "prevtxs" (string, optional) An json array of previous dependent transaction outputs

   ```
   [               (json array of json objects, or 'null' if none provided)
     {
       "txid":"id",             (string, required) The transaction id
       "vout":n,                  (numeric, required) The output number
       "scriptPubKey": "hex",   (string, required) script key
       "redeemScript": "hex",   (string, required for P2SH) redeem script
       "amount": value            (numeric, required) The amount spent
     }
     ,...
   ]
   ```

3. "privatekeys" (string, optional) A json array of base58-encoded private keys for signing

   ```
   [                  (json array of strings, or 'null' if none provided)
     "privatekey"   (string) private key in base58-encoding
     ,...
   ]
   ```

4. "sighashtype" (string, optional, default=ALL) The signature hash type. Must be one of

   ```
   "ALL"
   "NONE"
   "SINGLE"
   "ALL|ANYONECANPAY"
   "NONE|ANYONECANPAY"
   "SINGLE|ANYONECANPAY"
   ```

5. "branchid" (string, optional) The hex representation of the consensus branch id to sign with. This can be used to force signing with consensus rules that are ahead of the node's current height.

Result:

```
{
  "hex" : "value",           (string) The hex-encoded raw transaction with signature(s)
  "complete" : true|false,   (boolean) If the transaction has a complete set of signatures
  "errors" : [                 (json array of objects) Script verification errors (if there are any)
    {
      "txid" : "hash",           (string) The hash of the referenced, previous transaction
      "vout" : n,                (numeric) The index of the output to spent and used as input
      "scriptSig" : "hex",       (string) The hex-encoded signature script
      "sequence" : n,            (numeric) Script sequence number
      "error" : "text"           (string) Verification or signing error related to the input
    }
    ,...
  ]
}
```

Examples:

```
> verus signrawtransaction "myhex"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "signrawtransaction", "params": ["myhex"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

Util

`createmultisig nrequired ["key",...]`

Creates a multi-signature address with n signature of m keys required.
It returns a json object with the address and redeemScript.

Arguments:

1. nrequired (numeric, required) The number of required signatures out of the n keys or addresses.
2. "keys" (string, required) A json array of keys which are Komodo addresses or hex-encoded public keys

   ```
   [
     "key"    (string) Komodo address or hex-encoded public key
     ,...
   ]
   ```

   ### Result:

   ```
   {
   "address":"multisigaddress",  (string) The value of the new multisig address.
   "redeemScript":"script"       (string) The string value of the hex-encoded redemption script.
   }
   ```

   Examples:
   Create a multisig address from 2 addresses

   ```
   > verus createmultisig 2 "[\"RTZMZHDFSTFQst8XmX2dR4DaH87cEUs3gC\",\"RNKiEBduBru6Siv1cZRVhp4fkZNyPska6z\"]"
   ```

   As a json rpc call

   ```
   > curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "createmultisig", "params": [2, "[\"RTZMZHDFSTFQst8XmX2dR4DaH87cEUs3gC\",\"RNKiEBduBru6Siv1cZRVhp4fkZNyPska6z\"]"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
   ```

`estimatefee nblocks`

Estimates the approximate fee per kilobyte needed for a transaction to begin confirmation within nblocks blocks.

Arguments:

1. nblocks (numeric)

Result:

n : (numeric) estimated fee-per-kilobyte

minimum fee is returned if not enough transactions and blocks have been observed to make an estimate.

Example:

```
> verus estimatefee 6
```

`estimatepriority nblocks`

Estimates the approximate priority a zero-fee transaction needs to begin confirmation within nblocks blocks.

Arguments:

1. nblocks (numeric)

Result:

n : (numeric) estimated priority

-1.0 is returned if not enough transactions and
blocks have been observed to make an estimate.

Example:

```
> verus estimatepriority 6
```

`invalidateblock "hash"`

Permanently marks a block as invalid, as if it violated a consensus rule.

Arguments:

1. hash (string, required) the hash of the block to mark as invalid

`jumblr_deposit "depositaddress"`

`jumblr_pause`

`jumblr_resume`

`jumblr_secret "secretaddress"`

`reconsiderblock "hash"`

Removes invalidity status of a block and its descendants, reconsider them for activation.
This can be used to undo the effects of invalidateblock.

Arguments:

1. hash (string, required) the hash of the block to reconsider

Result:

Examples:

```
> verus invalidateblock "blockhash"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "invalidateblock", "param
s": ["blockhash"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`validateaddress "verusaddress"`

Return information about the given Verus address.

Arguments:

1. "verusaddress" (string, required) The Komodo address to validate

Result:

```
{
  "isvalid" : true|false,         (boolean) If the address is valid or not. If not, this is the only property ret
urned.
  "address" : "verusaddress",   (string) The Verus address validated
  "scriptPubKey" : "hex",       (string) The hex encoded scriptPubKey generated by the address
  "ismine" : true|false,          (boolean) If the address is yours or not
  "isscript" : true|false,        (boolean) If the key is a script
  "pubkey" : "publickeyhex",    (string) The hex value of the raw public key
  "iscompressed" : true|false,    (boolean) If the address is compressed
  "account" : "account"         (string) DEPRECATED. The account associated with the address, "" is the default a
ccount
}
```

Examples:

```
> verus validateaddress "RTZMZHDFSTFQst8XmX2dR4DaH87cEUs3gC"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "validateaddress", "param
s": ["RTZMZHDFSTFQst8XmX2dR4DaH87cEUs3gC"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_validateaddress "zaddr"`

Return information about the given z address.

Arguments:

1. "zaddr" (string, required) The z address to validate

Result:

```
{
  "isvalid" : true|false,      (boolean) If the address is valid or not. If not, this is the only property return
ed.
  "address" : "zaddr",         (string) The z address validated
  "type" : "xxxx",             (string) "sprout" or "sapling"
  "ismine" : true|false,       (boolean) If the address is yours or not
  "payingkey" : "hex",         (string) [sprout] The hex value of the paying key, a_pk
  "transmissionkey" : "hex",   (string) [sprout] The hex value of the transmission key, pk_enc
  "diversifier" : "hex",       (string) [sapling] The hex value of the diversifier, d
  "diversifiedtransmissionkey" : "hex", (string) [sapling] The hex value of pk_d
}
```

Examples:

```
> verus z_validateaddress "zcWsmqT4X2V4jgxbgiCzyrAfRT1vi1F4sn7M5Pkh66izzw8Uk7LBGAH3DtcSMJeUb2pi3W4SQF8LMKkU2cUuVP
68yAGcomL"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_validateaddress", "param
s": ["zcWsmqT4X2V4jgxbgiCzyrAfRT1vi1F4sn7M5Pkh66izzw8Uk7LBGAH3DtcSMJeUb2pi3W4SQF8LMKkU2cUuVP68yAGcomL"] }' -H 'co
ntent-type: text/plain;' http://127.0.0.1:27486/
```

VDXF

`getvdxfid "vdxfuri" '{"vdxfkey":"i-address or vdxfkey", "uint256":"hexstr", "indexnum":0}'`

Returns the VDXF key of the URI string. For example "vrsc::system.currency.export"

Arguments:

"vdxfuri" (string, required) This message is converted from hex, the data is hashed, then returned

```
{
  "vdxfkey":"i-address or vdxfkey"   (string, optional) VDXF key or i-address to combine via hash
  "uint256":"32bytehex"              (hexstr, optional) 256 bit hash to combine with hash
  "indexnum":int                       (integer, optional) int32_t number to combine with hash
}
```

Result:

```
{                                       (object) object with both base58check and hex vdxfid values of string an
d parents
  "vdxfid"                              (base58check) i-ID of the URI processed with the VDXF & all combined par
ameters
  "hash160result"                       (hexstring) 20 byte hash in hex of the URL string passed in, processed w
ith the VDXF
  "qualifiedname":                      (object) separate name and parent ID value
  {
    "name": "namestr"                   (string) leaf name
    "parentid" | "namespace":"string"   (string) parent ID (or namespace if VDXF key) of name
  }
  "bounddata": {                        (object) if additional data is bound to create the value, it is returned
here  {
    "vdxfkey":"i-address or vdxfkey"    (string) i-address that was combined via hash
    "uint256":"32bytehex"               (hexstr) 256 bit hash combined with hash
    "indexnum":int                      (integer) int32_t combined with hash
  }
}
```

Examples:

```
Create the signature
> verus getvdxfid "system.currency.export"

Verify the signature
> verus getvdxfid "idname::userdefinedgroup.subgroup.publishedname"

As json rpc
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getvdxfid", "params": ["id
name::userdefinedgroup.subgroup.publishedname"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

Wallet

`addmultisigaddress nrequired ["key",...] ( "account" )`

Add a nrequired-to-sign multisignature address to the wallet.
Each key is a VRSC address or hex-encoded public key.
If 'account' is specified (DEPRECATED), assign address to that account.

Arguments:

1. nrequired (numeric, required) The number of required signatures out of the n keys or addresses.
2. "keysobject" (string, required) A json array of VRSC addresses or hex-encoded public keys

```
[
  "address"  (string) VRSC address or hex-encoded public key
  ...,
]
```

3. "account" (string, optional) DEPRECATED. If provided, MUST be set to the empty string "" to represent the default account. Passing any other string will result in an error.

Result:

"VRSC_address" (string) A VRSC address associated with the keys.

Examples:

Add a multisig address from 2 addresses

```
> verus addmultisigaddress 2 "[\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\",\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\"]"
```

As json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "addmultisigaddress", "para
ms": [2, "[\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\",\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\"]"] }' -H 'content-type: tex
t/plain;' http://127.0.0.1:27486/
```

`backupwallet "destination"`

Safely copies `wallet.dat` to destination filename

Arguments:

1. "destination" (string, required) The destination filename, saved in the directory set by `-exportdir` option.

Result:

"path" (string) The full path of the destination file

Examples:

```
> verus backupwallet "backupdata"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "backupwallet", "params":
["backupdata"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`convertpassphrase "walletpassphrase"`

Converts Verus Desktop, Agama, Verus Agama, or Verus Mobile passphrase to a private key and WIF (for import with importprivkey).

Arguments:

1. "walletpassphrase" (string, required) Wallet passphrase

Result:

```
"walletpassphrase": "walletpassphrase", (string) Wallet passphrase you entered
"address": "verus address",            (string) Address corresponding to your passphrase
"pubkey": "publickeyhex",              (string) The hex value of the raw public key
"privkey": "privatekeyhex",            (string) The hex value of the raw private key
"wif": "wif"                           (string) The private key in WIF format to use with 'importprivkey'
```

Examples:

```
> verus convertpassphrase "walletpassphrase"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "convertpassphrase", "param
s": ["walletpassphrase"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`dumpprivkey "t-addr"`

Reveals the private key corresponding to 't-addr'.
Then the importprivkey can be used with this output

Arguments:

1. "t-addr" (string, required) The transparent address for the private key

Result:

"key" (string) The private key

Examples:

```
> verus dumpprivkey "myaddress"
> verus importprivkey "mykey"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "dumpprivkey", "params":
["myaddress"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`dumpwallet "filename" (omitemptytaddresses)`

Dumps taddr wallet keys in a human-readable format. Overwriting an existing file is not permitted.

Arguments:

1. "filename" (string, required) The filename, saved in folder set by verusd -exportdir option
2. "omitemptytaddresses" (boolean, optional) Defaults to false. If true, export only addresses with indexed UTXOs or that control IDs in the wallet (do not use this option without being sure that all addresses of interest are included)

Result:

"path" (string) The full path of the destination file

Examples:

```
> verus dumpwallet "test"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "dumpwallet", "params": ["t
est"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`encryptwallet "passphrase"`

> WARNING: encryptwallet is disabled.
> To enable it, restart verusd with the `-experimentalfeatures` and `-developerencryptwallet` commandline options, or add these two lines to the VRSC.conf file:

```
experimentalfeatures=1
developerencryptwallet=1
```

Encrypts the wallet with 'passphrase'. This is for first time encryption.
After this, any calls that interact with private keys such as sending or signing will require the passphrase to be set prior the making these calls.
Use the walletpassphrase call for this, and then walletlock call.
If the wallet is already encrypted, use the walletpassphrasechange call.
Note that this will shutdown the server.

Arguments:

1. "passphrase" (string) The pass phrase to encrypt the wallet with. It must be at least 1 character, but should be long.
   Examples:

Encrypt you wallet

```
> verus encryptwallet "my pass phrase"
```

Now set the passphrase to use the wallet, such as for signing or sending VRSC

```
> verus walletpassphrase "my pass phrase"
```

Now we can so something like sign

```
> verus signmessage "VRSC_address" "test message"
```

Now lock the wallet again by removing the passphrase

```
> verus walletlock
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "encryptwallet", "params":
["my pass phrase"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getaccount "VRSC_address"`

DEPRECATED. Returns the account associated with the given address.

Arguments:

1. "VRSC_address" (string, required) The VRSC address for account lookup.

Result:

"accountname" (string) the account address

Examples:

```
> verus getaccount "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getaccount", "params": ["R
D6GgnrMpPaTSMn8vai6yiGA7mN4QGPV"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getaccountaddress "account"`

DEPRECATED. Returns the current VRSC address for receiving payments to this account.

Arguments:

1. "account" (string, required) MUST be set to the empty string "" to represent the default account. Passing any other string will result in an error.

Result:

"VRSC_address" (string) The account VRSC address

Examples:

```
> verus getaccountaddress
> verus getaccountaddress ""
> verus getaccountaddress "myaccount"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getaccountaddress", "param
s": ["myaccount"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getaddressesbyaccount "account"`

DEPRECATED. Returns the list of addresses for the given account.

Arguments:

1. "account" (string, required) MUST be set to the empty string "" to represent the default account. Passing any other string will result in an error.

Result:

```
[                     (json array of string)
  "VRSC_address"  (string) a VRSC address associated with the given account
  ,...
]
```

Examples:

```
> verus getaddressesbyaccount "tabby"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getaddressesbyaccount", "p
arams": ["tabby"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getbalance ( "account" minconf includeWatchonly )`

Returns the server's total available balance.

Arguments:

1. "account" (string, optional) DEPRECATED. If provided, it MUST be set to the empty string "" or to the string "*", either of which will give the total available balance. Passing any other string will result in an error.
2. minconf (numeric, optional, default=1) Only include transactions confirmed at least this many times.
3. includeWatchonly (bool, optional, default=false) Also include balance in watchonly addresses (see 'importaddress')

Result:

amount (numeric) The total amount in VRSC received for this account.

Examples:

The total amount in the wallet

```
> verus getbalance
```

The total amount in the wallet at least 5 blocks confirmed

```
> verus getbalance "*" 6
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getbalance", "params":
["*", 6] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getcurrencybalance "address" ( minconf ) ( friendlynames ) (includeshared)`

Returns the balance in all currencies of a taddr, zaddr, iaddr or ID belonging to the node's wallet.

CAUTION: If the wallet has only an incoming viewing key for this address, then spends cannot be
detected, and so the returned balance may be larger than the actual balance.

Arguments:

1. "address" (string) The selected address. It may be a transparent, private or identity address and include `z\*`, `R\*`, and `i\*` wildcards.
2. minconf (numeric, optional, default=1) Only include transactions confirmed at least this many times.
3. friendlynames (boolean, optional, default=true) use friendly names instead of i-addresses.
4. includeshared (bool, optional, default=false) Include outputs that can also be spent by others

Result:

amount (numeric) The total amount in VRSC received for this address.

Examples:

The total amount received by address "myaddress"

```
> verus getcurrencybalance "myaddress"
```

The total amount received by address "myaddress" at least 5 blocks confirmed

```
> verus getcurrencybalance "myaddress" 5
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getcurrencybalance", "para
ms": ["myaddress", 5] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getnewaddress ( "account" )`

Returns a new VRSC address for receiving payments.

Arguments:

1. "account" (string, optional) DEPRECATED. If provided, it MUST be set to the empty string "" to represent the default account. Passing any other string will result in an error.

Result:

"VRSC_address" (string) The new VRSC address

Examples:

```
> verus getnewaddress
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getnewaddress", "params":
[] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getrawchangeaddress`

Returns a new VRSC address, for receiving change.
This is for use with raw transactions, NOT normal use.

Result:

"address" (string) The address

Examples:

```
> verus getrawchangeaddress
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getrawchangeaddress", "par
ams": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getreceivedbyaccount "account" ( minconf )`

DEPRECATED. Returns the total amount received by addresses with in transactions with at least [minconf] confirmations.

Arguments:

1. "account" (string, required) MUST be set to the empty string "" to represent the default account. Passing any other string will result in an error.
2. minconf (numeric, optional, default=1) Only include transactions confirmed at least this many times.

Result:

amount (numeric) The total amount in VRSC received for this account.

Examples:

Amount received by the default account with at least 1 confirmation

```
> verus getreceivedbyaccount ""
```

Amount received at the tabby account including unconfirmed amounts with zero confirmations

```
> verus getreceivedbyaccount "tabby" 0
```

The amount with at least 6 confirmation, very safe

```
> verus getreceivedbyaccount "tabby" 6
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getreceivedbyaccount", "pa
rams": ["tabby", 6] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`getreceivedbyaddress "VRSC_address" ( minconf )`

Returns the total amount received by the given VRSC address in transactions with at least minconf confirmations.

Arguments:

1. "VRSC_address" (string, required) The VRSC address for transactions.
2. minconf (numeric, optional, default=1) Only include transactions confirmed at least this many times.

Result:

amount (numeric) The total amount in VRSC received at this address.

Examples:

The amount from transactions with at least 1 confirmation

```
> verus getreceivedbyaddress "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV"
```

The amount including unconfirmed transactions, zero confirmations

```
> verus getreceivedbyaddress "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" 0
```

The amount with at least 6 confirmations, very safe

```
> verus getreceivedbyaddress "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" 6
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getreceivedbyaddress", "pa
rams": ["RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV", 6] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`gettransaction "txid" ( includeWatchonly )`

Get detailed information about in-wallet transaction

Arguments:

1. "txid" (string, required) The transaction id
2. "includeWatchonly" (bool, optional, default=false) Whether to include watchonly addresses in balance calculation and details[]

Result:

```
{
  "amount" : x.xxx,        (numeric) The transaction amount in VRSC
  "confirmations" : n,     (numeric) The number of confirmations
  "blockhash" : "hash",  (string) The block hash
  "blockindex" : xx,       (numeric) The block index
  "blocktime" : ttt,       (numeric) The time in seconds since epoch (1 Jan 1970 GMT)
  "txid" : "transactionid",   (string) The transaction id.
  "time" : ttt,            (numeric) The transaction time in seconds since epoch (1 Jan 1970 GMT)
  "timereceived" : ttt,    (numeric) The time received in seconds since epoch (1 Jan 1970 GMT)
  "details" : [
    {
      "account" : "accountname",  (string) DEPRECATED. The account name involved in the transaction, can be "" fo
r the default account.
      "address" : "VRSC_address",   (string) The VRSC address involved in the transaction
      "category" : "send|receive",    (string) The category, either 'send' or 'receive'
      "amount" : x.xxx                  (numeric) The amount in VRSC
      "vout" : n,                       (numeric) the vout value
    }
    ,...
  ],
  "vjoinsplit" : [
    {
      "anchor" : "treestateref",          (string) Merkle root of note commitment tree
      "nullifiers" : [ string, ... ]      (string) Nullifiers of input notes
      "commitments" : [ string, ... ]     (string) Note commitments for note outputs
      "macs" : [ string, ... ]            (string) Message authentication tags
      "vpub_old" : x.xxx                  (numeric) The amount removed from the transparent value pool
      "vpub_new" : x.xxx,                 (numeric) The amount added to the transparent value pool
    }
    ,...
  ],
  "hex" : "data"         (string) Raw data for transaction
}
```

Examples:

```
> verus gettransaction "1075db55d416d3ca199f55b6084e2115b9345e16c5cf302fc80e9d5fbf5d48d"
> verus gettransaction "1075db55d416d3ca199f55b6084e2115b9345e16c5cf302fc80e9d5fbf5d48d" true
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "gettransaction", "params":
["1075db55d416d3ca199f55b6084e2115b9345e16c5cf302fc80e9d5fbf5d48d"] }' -H 'content-type: text/plain;' http://127.
0.0.1:27486/
```

`getunconfirmedbalance`

Returns the server's total unconfirmed balance

`getwalletinfo`

Returns an object containing various wallet state info.

Result:

```
{
  "walletversion": xxxxx,     (numeric) the wallet version
  "balance": xxxxxxx,         (numeric) the total confirmed balance of the wallet in VRSC
  "reserve_balance": xxxxxxx, (numeric) for PBaaS reserve chains, the total confirmed reserve balance of the wall
et in VRSC
  "unconfirmed_balance": xxx, (numeric) the total unconfirmed balance of the wallet in VRSC
  "unconfirmed_reserve_balance": xxx, (numeric) total unconfirmed reserve balance of the wallet in VRSC
  "immature_balance": xxxxxx, (numeric) the total immature balance of the wallet in VRSC
  "immature_reserve_balance": xxxxxx, (numeric) total immature reserve balance of the wallet in VRSC
  "eligible_staking_balance": xxxxxx, (numeric) eligible staking balance in VRSC
  "reserve_balance": {"currency":XXXXXX,...} (json) json with currencies and the balances
  "txcount": xxxxxxx,         (numeric) the total number of transactions in the wallet
  "keypoololdest": xxxxxx,    (numeric) the timestamp (seconds since GMT epoch) of the oldest pre-generated key i
n the key pool
  "keypoolsize": xxxx,        (numeric) how many new keys are pre-generated
  "unlocked_until": ttt,      (numeric) the timestamp in seconds since epoch (midnight Jan 1 1970 GMT) that the w
allet is unlocked for transfers, or 0 if the wallet is locked
  "paytxfee": x.xxxx,         (numeric) the transaction fee configuration, set in VRSC/kB
  "seedfp": "uint256",        (string) the BLAKE2b-256 hash of the HD seed
}
```

Examples:

```
> verus getwalletinfo
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getwalletinfo", "params":
[] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`importaddress "address" ( "label" rescan )`

Adds an address or script (in hex) that can be watched as if it were in your wallet but cannot be used to spend.

Arguments:

1. "address" (string, required) The address
2. "label" (string, optional, default="") An optional label
3. rescan (boolean, optional, default=true) Rescan the wallet for transactions

> Note: This call can take minutes to complete if rescan is true.

Examples:

Import an address with rescan

```
> verus importaddress "myaddress"
```

Import using a label without rescan

```
> verus importaddress "myaddress" "testing" false
```

As a JSON-RPC call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "importaddress", "params":
["myaddress", "testing", false] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`importprivkey "verusprivkey" ( "label" rescan )`

Adds a private key (as returned by dumpprivkey) to your wallet.

Arguments:

1. "verusprivkey" (string, required) The private key (see dumpprivkey)
2. "label" (string, optional, default="") An optional label
3. rescan (boolean, optional, default=true) Rescan the wallet for transactions

> Note: This call can take minutes to complete if rescan is true.

Examples:

Dump a private key

```
> verus dumpprivkey "myaddress"
```

Import the private key with rescan

```
> verus importprivkey "mykey"
```

Import using a label and without rescan

```
> verus importprivkey "mykey" "testing" false
```

As a JSON-RPC call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "importprivkey", "params":
["mykey", "testing", false] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`importwallet "filename"`

Imports taddr keys from a wallet dump file (see dumpwallet).

Arguments:

1. "filename" (string, required) The wallet file

Examples:

Dump the wallet

```
> verus dumpwallet "nameofbackup"
```

Import the wallet

```
> verus importwallet "path/to/exportdir/nameofbackup"
```

Import using the json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "importwallet", "params":
["path/to/exportdir/nameofbackup"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`keypoolrefill ( newsize )`

Fills the keypool.

Arguments
1. newsize (numeric, optional, default=100) The new keypool size

Examples:

```
> verus keypoolrefill
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "keypoolrefill", "params":
[] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`listaccounts ( minconf includeWatchonly)`

DEPRECATED. Returns Object that has account names as keys, account balances as values.

Arguments:

1. minconf (numeric, optional, default=1) Only include transactions with at least this many confirmations
2. includeWatchonly (bool, optional, default=false) Include balances in watchonly addresses (see 'importaddress')

Result:

```
{                      (json object where keys are account names, and values are numeric balances
  "account": x.xxx,  (numeric) The property name is the account name, and the value is the total balance for the
account.
```

```
  ...
}
```

Examples:

List account balances where there at least 1 confirmation

```
> verus listaccounts
```

List account balances including zero confirmation transactions

```
> verus listaccounts 0
```

List account balances for 6 or more confirmations

```
> verus listaccounts 6
```

As json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "listaccounts", "params":
[6] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

listaddressgroupings

Lists groups of addresses which have had their common ownership
made public by common use as inputs or as the resulting change
in past transactions

Result:

```
[
  [
    [
      "VRSC address",     (string) The VRSC address
      amount,                 (numeric) The amount in VRSC
      "account"             (string, optional) The account (DEPRECATED)
    ]
    ,...
  ]
  ,...
]
```

Examples:

```
> verus listaddressgroupings
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "listaddressgroupings", "pa
rams": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

listlockunspent

Returns list of temporarily unspendable outputs.
See the lockunspent call to lock and unlock transactions for spending.

Result:

```
[
  {
    "txid" : "transactionid",     (string) The transaction id locked
    "vout" : n                      (numeric) The vout value
  }
  ,...
]
```

Examples:

List the unspent transactions

```
> verus listunspent
```

Lock an unspent transaction

```
> verus lockunspent false "[{\"txid\":\"a08e6907dbbd3d809776dbfc5d82e371b764ed838b5655e72f463568df1aadf0\",\"vout
\":1}]"
```

List the locked transactions

```
> verus listlockunspent
```

Unlock the transaction again

```
> verus lockunspent true "[{\"txid\":\"a08e6907dbbd3d809776dbfc5d82e371b764ed838b5655e72f463568df1aadf0\",\"vout
\":1}]"
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "listlockunspent", "param
s": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

listreceivedbyaccount (minconf includeempty includeWatchonly)

DEPRECATED. List balances by account.

Arguments:

1. minconf (numeric, optional, default=1) The minimum number of confirmations before payments are included.
2. includeempty (boolean, optional, default=false) Whether to include accounts that haven't received any payments.
3. includeWatchonly (bool, optional, default=false) Whether to include watchonly addresses (see 'importaddress').

Result:

```
[
  {
    "involvesWatchonly" : true,   (bool) Only returned if imported addresses were involved in transaction
    "account" : "accountname",  (string) The account name of the receiving account
    "amount" : x.xxx,             (numeric) The total amount received by addresses with this account
    "confirmations" : n           (numeric) The number of confirmations of the most recent transaction included
  }
  ,...
]
```

Examples:

```
> verus listreceivedbyaccount
> verus listreceivedbyaccount 6 true
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "listreceivedbyaccount", "p
arams": [6, true, true] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

listreceivedbyaddress (minconf includeempty includeWatchonly)

List balances by receiving address.

Arguments:

1. minconf (numeric, optional, default=1) The minimum number of confirmations before payments are included.
2. includeempty (numeric, optional, default=false) Whether to include addresses that haven't received any payments.
3. includeWatchonly (bool, optional, default=false) Whether to include watchonly addresses (see 'importaddress').

Result:

```
[
  {
    "involvesWatchonly" : true,        (bool) Only returned if imported addresses were involved in transaction
    "address" : "receivingaddress",    (string) The receiving address
    "account" : "accountname",         (string) DEPRECATED. The account of the receiving address. The default acc
ount is "".
    "amount" : x.xxx,                  (numeric) The total amount in VRSC received by the address
    "confirmations" : n                (numeric) The number of confirmations of the most recent transaction inclu
ded
  }
  ,...
]
```

Examples:

```
> verus listreceivedbyaddress
> verus listreceivedbyaddress 6 true
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "listreceivedbyaddress", "p
arams": [6, true, true] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

listsinceblock ("blockhash" target-confirmations includeWatchonly)

Get all transactions in blocks since block [blockhash], or all transactions if omitted

Arguments:

1. "blockhash" (string, optional) The block hash to list transactions since
2. target-confirmations: (numeric, optional) The confirmations required, must be 1 or more
3. includeWatchonly: (bool, optional, default=false) Include transactions to watchonly addresses (see 'importaddress')

Result:

```
{
"transactions": [
 "account":"accountname",       (string) DEPRECATED. The account name associated with the transaction. Will
be "" for the default account.
 "address":"VRSC_address",    (string) The VRSC address of the transaction. Not present for move transaction
s (category = move).
 "category":"send|receive",     (string) The transaction category. 'send' has negative amounts, 'receive' ha
s positive amounts.
 "amount": x.xxx,          (numeric) The amount in VRSC. This is negative for the 'send' category, and for t
he 'move' category for moves
                                          outbound. It is positive for the 'receive' category, and for the 'mov
e' category for inbound funds.
 "vout" : n,               (numeric) the vout value
 "fee": x.xxx,             (numeric) The amount of the fee in VRSC. This is negative and only available for
the 'send' category of transactions.
 "confirmations": n,       (numeric) The number of confirmations for the transaction. Available for 'send' a
nd 'receive' category of transactions.
 "blockhash": "hashvalue",     (string) The block hash containing the transaction. Available for 'send' and
'receive' category of transactions.
 "blockindex": n,          (numeric) The block index containing the transaction. Available for 'send' and 'r
eceive' category of transactions.
 "blocktime": xxx,         (numeric) The block time in seconds since epoch (1 Jan 1970 GMT).
 "txid": "transactionid",  (string) The transaction id. Available for 'send' and 'receive' category of trans
actions.
 "time": xxx,              (numeric) The transaction time in seconds since epoch (Jan 1 1970 GMT).
 "timereceived": xxx,      (numeric) The time received in seconds since epoch (Jan 1 1970 GMT). Available fo
r 'send' and 'receive' category of transactions.
 "comment": "...",       (string) If a comment is associated with the transaction.
 "to": "...",            (string) If a comment to is associated with the transaction.
],
"lastblock": "lastblockhash"     (string) The hash of the last block
}
```

Examples:

```
> verus listsinceblock
> verus listsinceblock "000000000000000bacf66f7497b7dc45ef753ee9a7d38571037cdb1a57f663ad" 6
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "listsinceblock", "params":
["000000000000000bacf66f7497b7dc45ef753ee9a7d38571037cdb1a57f663ad", 6] }' -H 'content-type: text/plain;' http://
127.0.0.1:27486/
```

listtransactions ("account" count from includeWatchonly)

Returns up to 'count' most recent transactions skipping the first 'from' transactions for account 'account'.

Arguments:

1. "account" (string, optional) DEPRECATED. The account name. Should be "*".
2. count (numeric, optional, default=10) The number of transactions to return
3. from (numeric, optional, default=0) The number of transactions to skip
4. includeWatchonly (bool, optional, default=false) Include transactions to watchonly addresses (see 'importaddress')

Result:

```
[
  {
    "account":"accountname",       (string) DEPRECATED. The account name associated with the transaction.
                                                It will be "" for the default account.
    "address":"VRSC_address",    (string) The VRSC address of the transaction. Not present for
                                                move transactions (category = move).
    "category":"send|receive|move", (string) The transaction category. 'move' is a local (off blockchain)
                                                transaction between accounts, and not associated with an address,
                                                transaction id or block. 'send' and 'receive' transactions are
                                                associated with an address, transaction id and block details
    "amount": x.xxx,          (numeric) The amount in VRSC. This is negative for the 'send' category, and for the
                                         'move' category for moves outbound. It is positive for the 'receive' cat
egory,
                                         and for the 'move' category for inbound funds.
    "vout" : n,               (numeric) the vout value
    "fee": x.xxx,             (numeric) The amount of the fee in VRSC. This is negative and only available for th
e
                                         'send' category of transactions.
    "confirmations": n,       (numeric) The number of confirmations for the transaction. Available for 'send' and
                                         'receive' category of transactions.
    "blockhash": "hashvalue", (string) The block hash containing the transaction. Available for 'send' and 'recei
ve'
                                          category of transactions.
    "blockindex": n,          (numeric) The block index containing the transaction. Available for 'send' and 'rec
eive'
                                          category of transactions.
    "txid": "transactionid", (string) The transaction id. Available for 'send' and 'receive' category of transact
ions.
    "time": xxx,              (numeric) The transaction time in seconds since epoch (midnight Jan 1 1970 GMT).
    "timereceived": xxx,      (numeric) The time received in seconds since epoch (midnight Jan 1 1970 GMT). Avail
able
                                          for 'send' and 'receive' category of transactions.
    "comment": "...",       (string) If a comment is associated with the transaction.
    "otheraccount": "accountname",  (string) For the 'move' category of transactions, the account the funds came
                                          from (for receiving funds, positive amounts), or went to (for sending f
unds,
                                          negative amounts).
    "size": n,                (numeric) Transaction size in bytes
  }
]
```

Examples:

List the most recent 10 transactions in the systems

```
> verus listtransactions
```

List transactions 100 to 120

```
> verus listtransactions "*" 20 100
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "listtransactions", "param
s": ["*", 20, 100] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

listunspent (minconf maxconf ["address",...])

Returns array of unspent transaction outputs with between minconf and maxconf (inclusive) confirmations.
Optionally filter to only include txouts paid to specified addresses.
Results are an array of Objects, each of which has:
{txid, vout, scriptPubKey, amount, confirmations}

Arguments:

1. minconf (numeric, optional, default=1) The minimum confirmations to filter
2. maxconf (numeric, optional, default=9999999) The maximum confirmations to filter
3. "addresses" (string) A json array of VRSC addresses to filter

```
[
  "address"   (string) VRSC address
  ,...
]
```

Result

```
[                   (array of json object)
{
 "txid" : "txid",          (string) the transaction id
 "vout" : n,               (numeric) the vout value
 "generated" : true|false  (boolean) true if txout is a coinbase transaction output
 "address" : "address",    (string) the Zcash address
 "account" : "account",    (string) DEPRECATED. The associated account, or "" for the default account
 "scriptPubKey" : "key",   (string) the script key
 "amount" : x.xxx,         (numeric) the transaction amount in VRSC
 "confirmations" : n,      (numeric) The number of confirmations
 "redeemScript" : n        (string) The redeemScript if scriptPubKey is P2SH
 "spendable" : xxx         (bool) Whether we have the private keys to spend this output
}
,...
]
```

Examples

```
> verus listunspent
> verus listunspent 6 9999999 "[\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\",\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\"]"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "listunspent", "params":
[6, 9999999 "[\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\",\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\"]"] }' -H 'content-type: t
ext/plain;' http://127.0.0.1:27486/
```

lockunspent unlock [{"txid":"txid","vout":n},...]

Updates list of temporarily unspendable outputs.
Temporarily lock (unlock=false) or unlock (unlock=true) specified transaction outputs.
A locked transaction output will not be chosen by automatic coin selection, when spending VRSC.
Locks are stored in memory only. Nodes start with zero locked outputs, and the locked output list is always cleared (by virtue of process exit) when a node stops or fails.
Also see the listunspent call

Arguments:

1. unlock (boolean, required) Whether to unlock (true) or lock (false) the specified transactions
2. "transactions" (string, required) A json array of objects. Each object the txid (string) vout (numeric)

```
[           (json array of json objects)
  {
    "txid":"id",    (string) The transaction id
    "vout": n         (numeric) The output number
  }
  ,...
]
```

Result:

true|false (boolean) Whether the command was successful or not

Examples:

List the unspent transactions

```
> verus listunspent
```

Lock an unspent transaction

```
> verus lockunspent false "[{\"txid\":\"a08e6907dbbd3d809776dbfc5d82e371b764ed838b5655e72f463568df1aadf0\",\"vout
\":1}]"
```

List the locked transactions

```
> verus listlockunspent
```

Unlock the transaction again

```
> verus lockunspent true "[{\"txid\":\"a08e6907dbbd3d809776dbfc5d82e371b764ed838b5655e72f463568df1aadf0\",\"vout
\":1}]"
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "lockunspent", "params": [f
alse, "[{\"txid\":\"a08e6907dbbd3d809776dbfc5d82e371b764ed838b5655e72f463568df1aadf0\",\"vout\":1}]"] }' -H 'cont
ent-type: text/plain;' http://127.0.0.1:27486/
```

`move "fromaccount" "toaccount" amount ( minconf "comment" )`

DEPRECATED. Move a specified amount from one account in your wallet to another.

Arguments:

1. "fromaccount" (string, required) MUST be set to the empty string "" to represent the default account. Passing any other string will result in an error.
2. "toaccount" (string, required) MUST be set to the empty string "" to represent the default account. Passing any other string will result in an error.
3. amount (numeric) Quantity of VRSC to move between accounts.
4. minconf (numeric, optional, default=1) Only use funds with at least this many confirmations.
5. "comment" (string, optional) An optional comment, stored in the wallet only.

Result:

true|false (boolean) true if successful.

Examples:

Move 0.01 VRSC from the default account to the account named tabby

```
> verus move "" "tabby" 0.01
```

Move 0.01 VRSC timotei to akiko with a comment and funds have 6 confirmations

```
> verus move "timotei" "akiko" 0.01 6 "happy birthday!"
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "move", "params": ["timote
i", "akiko", 0.01, 6, "happy birthday!"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`prunespentwallettransactions "txid"`

Remove all txs that are spent. You can clear all txs bar one, by specifiying a txid.

Please backup your wallet.dat before running this command.

Arguments:

1. "txid" (string, optional) The transaction id to keep.

Result:

```
{
  "total_transactions" : n,         (numeric) Transactions in wallet of VRSC
  "remaining_transactions" : n,     (numeric) Transactions in wallet after clean.
  "removed_transactions" : n,       (numeric) The number of transactions removed.
}
```

Examples:

```
> verus prunespentwallettransactions
> verus prunespentwallettransactions "1075db55d416d3ca199f55b6084e2115b9345e16c5cf302fc80e9d5fbf5d48d"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "prunespentwallettransactio
ns", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "prunespentwallettransactio
ns", "params": ["1075db55d416d3ca199f55b6084e2115b9345e16c5cf302fc80e9d5fbf5d48d"] }' -H 'content-type: text/plai
n;' http://127.0.0.1:27486/
```

`rescanfromheight (height)`

Rescans the current wallet from a specified height

Arguments:

1. "height" (int, optional) Defaults to 0, height to start rescanning from

Note: This call can take minutes or even hours to complete on very large wallets and rescans

Examples:

Initiate rescan of entire chain

```
> verus rescanfromheight
```

Initiate rescan from block 1000000

```
> verus rescanfromheight 1000000
```

`resendwallettransactions`

Immediately re-broadcast unconfirmed wallet transactions to all peers.
Intended only for testing; the wallet code periodically re-broadcasts automatically.
Returns array of transaction ids that were re-broadcast.

`sendfrom "fromaccount" "toVRSCaddress" amount ( minconf "comment" "comment-to" )`

DEPRECATED (use sendtoaddress). Sent an amount from an account to a VRSC address.
The amount is a real and is rounded to the nearest 0.00000001.

Arguments:

1. "fromaccount" (string, required) MUST be set to the empty string "" to represent the default account. Passing any other string will result in an error.
2. "toVRSCaddress" (string, required) The VRSC address to send funds to.
3. amount (numeric, required) The amount in VRSC (transaction fee is added on top).
4. minconf (numeric, optional, default=1) Only use funds with at least this many confirmations.
5. "comment" (string, optional) A comment used to store what the transaction is for.
   ```
                             This is not part of the transaction, just kept in your wallet.
   ```
6. "comment-to" (string, optional) An optional comment to store the name of the person or organization
   ```
                             to which you're sending the transaction. This is not part of the transaction,
                             it is just kept in your wallet.
   ```

Result:

"transactionid" (string) The transaction id.

Examples:

Send 0.01 VRSC from the default account to the address, must have at least 1 confirmation

```
> verus sendfrom "" "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" 0.01
```

Send 0.01 from the tabby account to the given address, funds must have at least 6 confirmations

```
> verus sendfrom "tabby" "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" 0.01 6 "donation" "seans outpost"
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "sendfrom", "params": ["tab
by", "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV", 0.01, 6, "donation", "seans outpost"] }' -H 'content-type: text/plain;'
http://127.0.0.1:27486/
```

`sendmany "fromaccount" {"address":amount,...} ( minconf "comment" ["address",...] )`

Send multiple times. Amounts are decimal numbers with at most 8 digits of precision.

Arguments:

1. "fromaccount" (string, required) MUST be set to the empty string "" to represent the default account. Passing any other string will result in an error.
2. "amounts" (string, required) A json object with addresses and amounts
   ```
   {
     "address":amount   (numeric) The VRSC address is the key, the numeric amount in VRSC is the value
     ,...
   }
   ```
3. minconf (numeric, optional, default=1) Only use the balance confirmed at least this many times.
4. "comment" (string, optional) A comment
5. subtractfeefromamount (string, optional) A json array with addresses. The fee will be equally deducted from the amount of each selected address. Those recipients will receive less VRSC than you enter in their corresponding amount field. If no addresses are specified here, the sender pays the fee.
   ```
   [
     "address"           (string) Subtract fee from this address
     ,...
   ]
   ```

Result:

"transactionid" (string) The transaction id for the send. Only 1 transaction is created regardless of the number of addresses.

Examples:

Send two amounts to two different addresses:

```
> verus sendmany "" "{\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\":0.01,\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\":0.02}"
```

Send two amounts to two different addresses setting the confirmation and comment:

```
> verus sendmany "" "{\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\":0.01,\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\":0.02}" 6 "te
sting"
```

Send two amounts to two different addresses, subtract fee from amount:

```
> verus sendmany "" "{\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\":0.01,\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\":0.02}" 1 ""
"[\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\",\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\"]"
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "sendmany", "params": ["",
"{\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\":0.01,\"RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV\":0.02}", 6, "testing"] }' -H 'con
tent-type: text/plain;' http://127.0.0.1:27486/
```

`sendtoaddress "VRSC_address" amount ( "comment" "comment-to" subtractfeefromamount )`

Send an amount to a given address. The amount is a real and is rounded to the nearest 0.00000001

Arguments:

1. "VRSC_address" (string, required) The VRSC address to send to.
2. "amount" (numeric, required) The amount in VRSC to send. eg 0.1
3. "comment" (string, optional) A comment used to store what the transaction is for. This is not part of the transaction, just kept in your wallet.
4. "comment-to" (string, optional) A comment to store the name of the person or organization to which you're sending the transaction. This is not part of the transaction, just kept in your wallet.
5. subtractfeefromamount (boolean, optional, default=false) The fee will be deducted from the amount being sent. The recipient will receive less VRSC than you enter in the amount field.

Result:

"transactionid" (string) The transaction id.

Examples:

```
> verus sendtoaddress "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" 0.1
> verus sendtoaddress "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" 0.1 "donation" "seans outpost"
> verus sendtoaddress "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" 0.1 "" "" true
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "sendtoaddress", "params":
["RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV", 0.1, "donation", "seans outpost"] }' -H 'content-type: text/plain;' http://1
27.0.0.1:27486/
```

`setaccount "VRSC_address" "account"`

DEPRECATED. Sets the account associated with the given address.

Arguments:

1. "VRSC_address" (string, required) The VRSC address to be associated with an account.
2. "account" (string, required) MUST be set to the empty string "" to represent the default account. Passing any other string will result in an error.

Examples:

```
> verus setaccount "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" "tabby"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "setaccount", "params": ["R
D6GgnrMpPaTSMn8vai6yiGA7mN4QGPV", "tabby"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`settxfee amount`

Set the transaction fee per kB.

Arguments:

1. amount (numeric, required) The transaction fee in VRSC/kB rounded to the nearest 0.00000001

Result:

true|false (boolean) Returns true if successful

Examples:

```
> verus settxfee 0.00001
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "settxfee", "params": [0.00
001] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_exportkey "zaddr"`

Reveals the zkey corresponding to 'zaddr'.
Then the `z_importkey` can be used with this output

Arguments:

1. "zaddr" (string, required) The zaddr for the private key

Result:

"key" (string) The private key

Examples:

```
> verus z_exportkey "myaddress"
> verus z_importkey "mykey"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_exportkey", "params":
["myaddress"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_exportviewingkey "zaddr"`

Reveals the viewing key corresponding to 'zaddr'.
Then the `z_importviewingkey` can be used with this output

Arguments:

1. "zaddr" (string, required) The zaddr for the viewing key

Result:
"vkey" (string) The viewing key

Examples:

```
> verus z_exportviewingkey "myaddress"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_exportviewingkey", "para
ms": ["myaddress"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_exportwallet "filename" (omitemptytaddresses)`

Exports all wallet keys, for taddr and zaddr, in a human-readable format. Overwriting an existing file is not permitted.

Arguments:

1. "filename" (string, required) The filename, saved in folder set by verusd -exportdir option
2. "omitemptytaddresses" (boolean, optional) Defaults to false. If true, export only addresses with indexed UTXOs or that control IDs in the wallet
   ```
                   (do not use this option without being sure that all addresses of interest are included)
   ```

Result:

"path" (string) The full path of the destination file

Examples:

```
> verus z_exportwallet "test"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_exportwallet", "params":
["test"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_getbalance "address" ( minconf )`

Returns the balance of a taddr or zaddr belonging to the node's wallet.

CAUTION: If the wallet has only an incoming viewing key for this address, then spends cannot be
detected, and so the returned balance may be larger than the actual balance.

Arguments:

1. "address" (string) The selected address. It may be a transparent or private address and
   ```
               include "z\*" "R\*" and "i\*" wildcards. Also `"idname@:private"` can be used
               to get the balance of a private address attached to an ID.
   ```
2. minconf (numeric, optional, default=1) Only include transactions confirmed at least this many times.

Result:

amount (numeric) The total amount in KMD received for this address.

Examples:

The total amount received by address "myaddress"

```
> verus z_getbalance "myaddress"
```

The total amount received by address "myaddress" at least 5 blocks confirmed

```
> verus z_getbalance "myaddress" 5
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_getbalance", "params":
["myaddress", 5] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_getnewaddress ( type )`

Returns a new shielded address for receiving payments.

With no arguments, returns a Sapling address.

Arguments:

1. "type" (string, optional, default="sapling") The type of address. One of ["sprout", "sapling"].

Result:

"VRSC_address" (string) The new shielded address.

Examples:

```
> verus z_getnewaddress
> verus z_getnewaddress sapling
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_getnewaddress", "param
```

```
s": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_getoperationresult '["operationid", ... ]'`

Retrieve the result and status of an operation which has finished, and then remove the operation from memory.

Arguments:

1. "operationid" (array, optional) A list of operation ids we are interested in. If not provided, examine all operations known to the node.

Result:

" [object, ...]" (array) A list of JSON objects

Examples:

```
> verus z_getoperationresult '["operationid", ... ]'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_getoperationresult", "pa
rams": ['["operationid", ... ]'] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_getoperationstatus '["operationid", ... ]'`

Get operation status and any associated result or error data. The operation will remain in memory.

Arguments:

1. "operationid" (array, optional) A list of operation ids we are interested in. If not provided, examine all operations known to the node.

Result:

" [object, ...]" (array) A list of JSON objects

Examples:

```
> verus z_getoperationstatus '["operationid", ... ]'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_getoperationstatus", "pa
rams": ['["operationid", ... ]'] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_gettotalbalance ( minconf includeWatchonly )`

Return the total value of funds stored in the node's wallet.

CAUTION: If the wallet contains any addresses for which it only has incoming viewing keys,
the returned private balance may be larger than the actual balance, because spends cannot
be detected with incoming viewing keys.

Arguments:

1. minconf (numeric, optional, default=1) Only include private and transparent transactions confirmed at least this many times.
2. includeWatchonly (bool, optional, default=false) Also include balance in watchonly addresses (see 'importaddress' and 'z_importviewingkey')

Result:

```
{
  "transparent": xxxxx,     (numeric) the total balance of transparent funds
  "private": xxxxx,         (numeric) the total balance of shielded funds (in both Sprout and Sapling addresses)
  "total": xxxxx,           (numeric) the total balance of both transparent and shielded funds
}
```

Examples:

The total amount in the wallet

```
> verus z_gettotalbalance
```

The total amount in the wallet at least 5 blocks confirmed

```
> verus z_gettotalbalance 5
```

As a json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_gettotalbalance", "param
s": [5] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_importkey "zkey" ( rescan startHeight )`

Adds a zkey (as returned by z_exportkey) to your wallet.

Arguments:

1. "zkey" (string, required) The zkey (see z_exportkey)
2. rescan (string, optional, default="whenkeyisnew") Rescan the wallet for transactions - can be "yes", "no" or "whenkeyisnew"
3. startHeight (numeric, optional, default=0) Block height to start rescan from

Note: This call can take minutes to complete if rescan is true.

Examples:

Export a zkey

```
> verus z_exportkey "myaddress"
```

Import the zkey with rescan

```
> verus z_importkey "mykey"
```

Import the zkey with partial rescan

```
> verus z_importkey "mykey" whenkeyisnew 30000
```

Re-import the zkey with longer partial rescan

```
> verus z_importkey "mykey" yes 200000
```

As a JSON-RPC call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_importkey", "params":
["mykey", "no"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_importviewingkey "vkey" ( rescan startHeight )`

Adds a viewing key (as returned by z_exportviewingkey) to your wallet.

Arguments:

1. "vkey" (string, required) The viewing key (see z_exportviewingkey)
2. rescan (string, optional, default="whenkeyisnew") Rescan the wallet for transactions - can be "yes", "no" or "whenkeyisnew"
3. startHeight (numeric, optional, default=0) Block height to start rescan from
4. zaddr (string, optional, default="") zaddr in case of importing viewing key for Sapling

Note: This call can take minutes to complete if rescan is true.

Examples:

Import a viewing key

```
> verus z_importviewingkey "vkey"
```

Import the viewing key without rescan

```
> verus z_importviewingkey "vkey", no
```

Import the viewing key with partial rescan

```
> verus z_importviewingkey "vkey" whenkeyisnew 30000
```

Re-import the viewing key with longer partial rescan

```
> verus z_importviewingkey "vkey" yes 20000
```

Import the viewing key for Sapling address

```
> verus z_importviewingkey "vkey" no 0 "zaddr"
```

As a JSON-RPC call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_importviewingkey", "para
ms": ["vkey", "no"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_importwallet "filename"`

Imports taddr and zaddr keys from a wallet export file (see z_exportwallet).

Arguments:

1. "filename" (string, required) The wallet file

Examples:

Dump the wallet

```
> verus z_exportwallet "nameofbackup"
```

Import the wallet

```
> verus z_importwallet "path/to/exportdir/nameofbackup"
```

Import using the json rpc call

```
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_importwallet", "params":
["path/to/exportdir/nameofbackup"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_listaddresses ( includeWatchonly )`

Returns the list of Sprout and Sapling shielded addresses belonging to the wallet.

Arguments:

1. includeWatchonly (bool, optional, default=false) Also include watchonly addresses (see 'z_importviewingkey')

Result:

```
[                     (json array of string)
  "zaddr"           (string) a zaddr belonging to the wallet
  ,...
]
```

Examples:

```
> verus z_listaddresses
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_listaddresses", "param
s": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_listoperationids`

Returns the list of operation ids currently known to the wallet.

Arguments:

1. "status" (string, optional) Filter result by the operation's state e.g. "success".

Result:

```
[                     (json array of string)
  "operationid"       (string) an operation id belonging to the wallet
  ,...
]
```

Examples:

```
> verus z_listoperationids
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_listoperationids", "para
ms": [] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_listreceivedbyaddress "address" ( minconf )`

Return a list of amounts received by a zaddr belonging to the node's wallet.

Arguments:

1. "address" (string) The private address.
2. minconf (numeric, optional, default=1) Only include transactions confirmed at least this many times.

Result:

```
{
  "txid": "txid",           (string) the transaction id
  "amount": xxxxx,         (numeric) the amount of value in the note
  "memo": xxxxx,           (string) hexadecimal string representation of memo field
  "jsindex" (sprout) : n,     (numeric) the joinsplit index
  "jsoutindex" (sprout) : n,     (numeric) the output index of the joinsplit
  "outindex" (sapling) : n,     (numeric) the output index
  "change": true|false,    (boolean) true if the address that received the note is also one of the sending addres
ses
}
```

Examples:

```
> verus z_listreceivedbyaddress "ztfaW34Gj9FrnGUEf833ywDVL62NWXBM81u6EQnM6VR45eYnXhwztecW1SjxA7JrmAXKJhxhj3vDNEpV
CQoSvVoSpmbhtjf"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_listreceivedbyaddress",
"params": ["ztfaW34Gj9FrnGUEf833ywDVL62NWXBM81u6EQnM6VR45eYnXhwztecW1SjxA7JrmAXKJhxhj3vDNEpVCQoSvVoSpmbhtjf"] }'
-H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_listunspent ( minconf maxconf includeWatchonly ["zaddr",...] )`

Returns array of unspent shielded notes with between minconf and maxconf (inclusive) confirmations.
Optionally filter to only include notes sent to specified addresses.
When minconf is 0, unspent notes with zero confirmations are returned, even though they are not immediately spendable.
Results are an array of Objects, each of which has:
{txid, jsindex, jsoutindex, confirmations, address, amount, memo} (Sprout)
{txid, outindex, confirmations, address, amount, memo} (Sapling)

Arguments:

1. minconf (numeric, optional, default=1) The minimum confirmations to filter
2. maxconf (numeric, optional, default=9999999) The maximum confirmations to filter
3. includeWatchonly (bool, optional, default=false) Also include watchonly addresses (see 'z_importviewingkey')
4. "addresses" (string) A json array of zaddrs (both Sprout and Sapling) to filter on. Duplicate addresses not allowed.

   ```
   [
     "address"     (string) zaddr
     ,...
   ]
   ```

   Result

   ```
   [                             (array of json object)
   {
    "txid" : "txid",          (string) the transaction id
    "jsindex" (sprout) : n,       (numeric) the joinsplit index
    "jsoutindex" (sprout) : n,       (numeric) the output index of the joinsplit
    "outindex" (sapling) : n,       (numeric) the output index
    "confirmations" : n,       (numeric) the number of confirmations
    "spendable" : true|false,  (boolean) true if note can be spent by wallet, false if address is watchonly
    "address" : "address",    (string) the shielded address
    "amount": xxxxx,          (numeric) the amount of value in the note
    "memo": xxxxx,            (string) hexademical string representation of memo field
    "change": true|false,     (boolean) true if the address that received the note is also one of the sending a
   ddresses
   }
   ,...
   ]
   ```

Examples

```
> verus z_listunspent
> verus z_listunspent 6 9999999 false "[\"ztbx5DLDxa5ZLFTchHhoPNkKs57QzSyib6UqXpEdy76T1aUdFxJt1w9318Z8DJ73XzbnWHK
EZP9Yjg712N5kMmP4QzS9iC9\",\"ztfaW34Gj9FrnGUEf833ywDVL62NWXBM81u6EQnM6VR45eYnXhwztecW1SjxA7JrmAXKJhxhj3vDNEpVCQoS
vVoSpmbhtjf\"]"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_listunspent", "params":
[6 9999999 false "[\"ztbx5DLDxa5ZLFTchHhoPNkKs57QzSyib6UqXpEdy76T1aUdFxJt1w9318Z8DJ73XzbnWHKEZP9Yjg712N5kMmP4QzS9
iC9\",\"ztfaW34Gj9FrnGUEf833ywDVL62NWXBM81u6EQnM6VR45eYnXhwztecW1SjxA7JrmAXKJhxhj3vDNEpVCQoSvVoSpmbhtjf\"]"] }' -
H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_mergetoaddress ["fromaddress", ... ] "toaddress" ( fee ) ( transparent_limit ) ( shielded_limit ) ( memo )`

WARNING: z_mergetoaddress is disabled.
To enable it, restart verusd with the `-experimentalfeatures` and
`-zmergetoaddress` commandline options, or add these two lines
to the `VRSC.conf` file:

```
experimentalfeatures=1
zmergetoaddress=1
```

Merge multiple UTXOs and notes into a single UTXO or note. Protected coinbase UTXOs are ignored, use `z_shieldcoinbase` to combine those into a single note.

This is an asynchronous operation, and UTXOs selected for merging will be locked. If there is an error, they are unlocked. The RPC call `listlockunspent` can be used to return a list of locked UTXOs.

The number of UTXOs and notes selected for merging can be limited by the caller. If the transparent limit parameter is set to zero, and Overwinter is not yet active, the `-mempooltxinputlimit` option will determine the number of UTXOs. After Overwinter has activated `-mempooltxinputlimit` is ignored and having a transparent input limit of zero will mean limit the number of UTXOs based on the size of the transaction. Any limit is constrained by the consensus rule defining a maximum transaction size of 100000 bytes before Sapling, and 2000000 bytes once Sapling activates.

Arguments:

1. fromaddresses (array, required) A JSON array with addresses.

   ```
                   The following special strings are accepted inside the array:
                       - "ANY_TADDR":   Merge UTXOs from any t-addrs belonging to the wallet.
                       - "ANY_SPROUT":  Merge notes from any Sprout zaddrs belonging to the wallet.
                       - "ANY_SAPLING": Merge notes from any Sapling zaddrs belonging to the wallet.
                   While it is possible to use a variety of different combinations of addresses and the above
   values,
                   it is not possible to send funds from both sprout and sapling addresses simultaneously. If
   a special
                   string is given, any given addresses of that type will be counted as duplicates and cause
   an error.
   ```

   ```
   [
     "address"          (string) Can be a t-addr or a zaddr
     ,...
   ]
   ```

2. "toaddress" (string, required) The t-addr or zaddr to send the funds to.
3. fee (numeric, optional, default=0.0001) The fee amount to attach to this transaction.
4. transparent_limit (numeric, optional, default=50) Limit on the maximum number of UTXOs to merge. Set to 0 to use node option -mempooltxinputlimit (before Overwinter), or as many as will fit in the transaction (after Overwinter).
5. shielded_limit (numeric, optional, default=20 Sprout or 200 Sapling Notes) Limit on the maximum number of notes to merge. Set to 0 to merge as many as will fit in the transaction.
6. "memo" (string, optional) Encoded as hex. When toaddress is a zaddr, this will be stored in the memo field of the new note.

Result:

```
{
  "remainingUTXOs": xxx               (numeric) Number of UTXOs still available for merging.
  "remainingTransparentValue": xxx    (numeric) Value of UTXOs still available for merging.
  "remainingNotes": xxx               (numeric) Number of notes still available for merging.
```

```
  "remainingShieldedValue": xxx       (numeric) Value of notes still available for merging.
  "mergingUTXOs": xxx                 (numeric) Number of UTXOs being merged.
  "mergingTransparentValue": xxx      (numeric) Value of UTXOs being merged.
  "mergingNotes": xxx                 (numeric) Number of notes being merged.
  "mergingShieldedValue": xxx         (numeric) Value of notes being merged.
  "opid": xxx                         (string) An operationid to pass to z_getoperationstatus to get the result o
f the operation.
}
```

Examples

```
> verus z_mergetoaddress '["ANY_SAPLING", "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV"]' ztfaW34Gj9FrnGUEf833ywDVL62NWXBM81
u6EQnM6VR45eYnXhwztecW1SjxA7JrmAXKJhxhj3vDNEpVCQoSvVoSpmbhtjf
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_mergetoaddress", "param
s": [["ANY_SAPLING", "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV"], "ztfaW34Gj9FrnGUEf833ywDVL62NWXBM81u6EQnM6VR45eYnXhwzte
cW1SjxA7JrmAXKJhxhj3vDNEpVCQoSvVoSpmbhtjf"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_sendmany "fromaddress" [{"address":... ,"amount":...},...] ( minconf ) ( fee )`

Send multiple times. Amounts are decimal numbers with at most 8 digits of precision.
Change generated from a taddr flows to a new taddr address, while change generated from a zaddr returns to itself.
When sending coinbase UTXOs to a zaddr, change is not allowed. The entire value of the UTXO(s) must be consumed.
Before Sapling activates, the maximum number of zaddr outputs is 54 due to transaction size limits.

Arguments:

1. "fromaddress" (string, required) The taddr or zaddr to send the funds from. Also `"idname@:private"` can be used

```
                    to get the source from a private address attached to an ID.
```

2. "amounts" (array, required) An array of json objects representing the amounts to send.

```
 [{
   "address":address  (string, required) The address is a taddr or zaddr. Also `"idname@:private"` can be us
ed
                      to send to a private address attached to an ID.
   "amount":amount    (numeric, required) The numeric amount in KMD is the value
   "memo":memo        (string, optional) If the address is a zaddr, raw data represented in hexadecimal stri
ng format
 }, ... ]
```

3. minconf (numeric, optional, default=1) Only use funds confirmed at least this many times.
4. fee (numeric, optional, default=0.0001) The fee amount to attach to this transaction.

Result:

"operationid" (string) An operationid to pass to z_getoperationstatus to get the result of the operation.

Examples:

```
> verus z_sendmany "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" '[{"address": "ztfaW34Gj9FrnGUEf833ywDVL62NWXBM81u6EQnM6VR4
5eYnXhwztecW1SjxA7JrmAXKJhxhj3vDNEpVCQoSvVoSpmbhtjf" ,"amount": 5.0}]'
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_sendmany", "params": ["R
D6GgnrMpPaTSMn8vai6yiGA7mN4QGPV", [{"address": "ztfaW34Gj9FrnGUEf833ywDVL62NWXBM81u6EQnM6VR45eYnXhwztecW1SjxA7Jrm
AXKJhxhj3vDNEpVCQoSvVoSpmbhtjf" ,"amount": 5.0}]] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_shieldcoinbase "fromaddress" "tozaddress" ( fee ) ( limit )`

Shield transparent coinbase funds by sending to a shielded zaddr. This is an asynchronous operation and utxos selected for shielding will be locked. If there is an error, they are unlocked. The RPC call `listlockunspent` can be used to return a list of locked utxos. The number of coinbase utxos selected for shielding can be limited by the caller. If the limit parameter is set to zero, and Overwinter is not yet active, the `-mempooltxinputlimit` option will determine the number of uxtos. Any limit is constrained by the consensus rule defining a maximum transaction size of 100000 bytes before Sapling, and 2000000 bytes once Sapling activates.

Arguments:

1. "fromaddress" (string, required) The address is a taddr or "`*`" for all taddrs belonging to the wallet.
2. "toaddress" (string, required) The address is a zaddr.
3. fee (numeric, optional, default=0.0001) The fee amount to attach to this transaction.
4. limit (numeric, optional, default=50) Limit on the maximum number of utxos to shield. Set to 0 to use node option -mempooltxinputlimit (before Overwinter), or as many as will fit in the transaction (after Overwinter).

Result:

```
{
  "remainingUTXOs": xxx       (numeric) Number of coinbase utxos still available for shielding.
  "remainingValue": xxx       (numeric) Value of coinbase utxos still available for shielding.
  "shieldingUTXOs": xxx        (numeric) Number of coinbase utxos being shielded.
  "shieldingValue": xxx        (numeric) Value of coinbase utxos being shielded.
  "opid": xxx          (string) An operationid to pass to z_getoperationstatus to get the result of the operatio
n.
}
```

Examples:

```
> verus z_shieldcoinbase "RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV" "ztfaW34Gj9FrnGUEf833ywDVL62NWXBM81u6EQnM6VR45eYnXhwz
tecW1SjxA7JrmAXKJhxhj3vDNEpVCQoSvVoSpmbhtjf"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_shieldcoinbase", "param
s": ["RD6GgnrMpPaTSMn8vai6yiGA7mN4QGPV", "ztfaW34Gj9FrnGUEf833ywDVL62NWXBM81u6EQnM6VR45eYnXhwztecW1SjxA7JrmAXKJhx
hj3vDNEpVCQoSvVoSpmbhtjf"] }' -H 'content-type: text/plain;' http://127.0.0.1:27486/
```

`z_viewtransaction "txid"`

Get detailed shielded information about in-wallet transaction

Arguments:

1. "txid" (string, required) The transaction id

Result:

```
{
  "txid" : "transactionid",   (string) The transaction id
  "spends" : [
    {
      "type" : "sprout|sapling",      (string) The type of address
      "js" : n,                       (numeric, sprout) the index of the JSDescription within vJoinSplit
      "jsSpend" : n,                  (numeric, sprout) the index of the spend within the JSDescription
      "spend" : n,                    (numeric, sapling) the index of the spend within vShieldedSpend
      "txidPrev" : "transactionid",   (string) The id for the transaction this note was created in
      "jsPrev" : n,                   (numeric, sprout) the index of the JSDescription within vJoinSplit
      "jsOutputPrev" : n,             (numeric, sprout) the index of the output within the JSDescription
      "outputPrev" : n,               (numeric, sapling) the index of the output within the vShieldedOutput
      "address" : "zcashaddress",     (string) The Zcash address involved in the transaction
      "value" : x.xxx                 (numeric) The amount in VRSC
      "valueZat" : xxxx               (numeric) The amount in zatoshis
    }
    ,...
  ],
  "outputs" : [
    {
      "type" : "sprout|sapling",      (string) The type of address
      "js" : n,                       (numeric, sprout) the index of the JSDescription within vJoinSplit
      "jsOutput" : n,                 (numeric, sprout) the index of the output within the JSDescription
      "output" : n,                   (numeric, sapling) the index of the output within the vShieldedOutput
      "address" : "zcashaddress",     (string) The Zcash address involved in the transaction
      "recovered" : true|false        (boolean, sapling) True if the output is not for an address in the wallet
      "value" : x.xxx                 (numeric) The amount in VRSC
      "valueZat" : xxxx               (numeric) The amount in zatoshis
      "memo" : "hexmemo",             (string) Hexademical string representation of the memo field
      "memoStr" : "memo",             (string) Only returned if memo contains valid UTF-8 text.
    }
    ,...
  ],
}
```

Examples:

```
> verus z_viewtransaction "1075db55d416d3ca199f55b6084e2115b9345e16c5cf302fc80e9d5fbf5d48d"
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "z_viewtransaction", "param
s": ["1075db55d416d3ca199f55b6084e2115b9345e16c5cf302fc80e9d5fbf5d48d"] }' -H 'content-type: text/plain;' http://
127.0.0.1:27486/
```

`zcbenchmark benchmarktype samplecount`

Runs a benchmark of the selected type samplecount times,
returning the running times of each sample.

Result:

```
[
  {
    "runningtime": runningtime
  },
  {
    "runningtime": runningtime
  }
  ...
]
```

`zcrawjoinsplit rawtx inputs outputs vpub_old vpub_new`

> inputs: a JSON object mapping {note: zcsecretkey, ...}
> outputs: a JSON object mapping {zcaddr: value, ...}

DEPRECATED. Splices a joinsplit into rawtx. Inputs are unilaterally confidential.
Outputs are confidential between sender/receiver. The vpub_old and vpub_new values are globally public and move transparent value into or out of the confidential value store, respectively.

> Note: The caller is responsible for delivering the output enc1 and enc2 to the appropriate recipients, as well as signing rawtxout and ensuring it is mined. (A future RPC call will deliver the confidential payments in-band on the blockchain.)

Output:

```
{
"encryptednote1": enc1,
"encryptednote2": enc2,
"rawtxn": rawtxout
}
```

`zcrawkeygen`

DEPRECATED. Generate a zcaddr which can send and receive confidential values.

Output:

```
{
  "zcaddress": zcaddr,
  "zcsecretkey": zcsecretkey,
  "zcviewingkey": zcviewingkey,
}
```

`zcrawreceive zcsecretkey encryptednote`

DEPRECATED. Decrypts encryptednote and checks if the coin commitments
are in the blockchain as indicated by the "exists" result.

Output:

```
{
  "amount": value,
  "note": noteplaintext,
  "exists": exists
}
```

`zcsamplejoinsplit`

Perform a joinsplit and return the JSDescription.

compiled by Oink.vrsc@, additions by Mike@, grewalsatinder@ and allbits@

> Note: last revision date 2024-01-30.