Electra RPC Commands and arguments

All command-line options (except for `-datadir` and `-conf`) may be specified in a configuration file, and all configuration file options may also be specified on the command line. Command-line options override values set in the configuration file. The configuration file is a list of `setting=value` pairs, one per line, with optional comments starting with the `#` character.

The configuration file is not automatically created; you can create it using your favorite plain-text editor. By default, electra-qt (or Electrad) will look for a file named `Electra.conf` in the electra data directory, but both the data directory and the configuration file path may be changed using the -datadir and -conf command-line arguments.

Platform	Path to data folder	Typical path to configuration file
Linux	~/	/home/username/.Electra/Electra.conf
macOS	~/Library/Application Support/	/Users/username/Library/Application Support/Eletra/electra.conf
Windows	%APPDATA%	(Vista-10) C:\Users\username\AppData\Roaming\Electra\Electra.conf (2000-XP) C:\Documents and Settings\username\Application Data\Electra\Electra.conf

Note: if running Electra in testnet mode, the sub-folder `testnet` will be appended to the data directory automatically.

Command line arguments

These commands are accurate as of Electra Core version 2.0.0

- electrad
- electra-cli
- electra-tx
- electra-qt

Electrad

Electra Core Daemon

Usage

Electrad [options]

Start Electra Core Daemon

Options

--help	This help message
--version	Print version and exit
--alerts	Receive and display P2P network alerts (default: 1)
--alertnotify=<*cmd*>	
	Execute command when a relevant alert is received or we see a really long fork (%s in cmd is replaced by message)
--blocknotify=<*cmd*>	
	Execute command when the best block changes (%s in cmd is replaced by block hash)
--blocksizenotify=<*cmd*>	
	Execute command when the best block changes and its size is over (%s in cmd is replaced by block hash, %d with the block size)
--checkblocks=<n>	
	How many blocks to check at startup (default: 500, 0 = all)
--conf=<*file*>	Specify configuration file (default: Electra.conf)
--daemon	Run in the background as a daemon and accept commands
--datadir=<*dir*>	
	Specify data directory
--dbcache=<*n*>	Set database cache size in megabytes (4 to 16384, default: 100)
--loadblock=<*file*>	
	Imports blocks from external blk000??.dat file on startup

--maxreorg=<n>	
	Set the Maximum reorg depth (default: 100)
--maxorphantx=<*n*>	
	Keep at most <n> unconnectable transactions in memory (default: 100)
--par=<*n*>	Set the number of script verification threads (-1 to 16, 0 = auto, <0 = leave that many cores free, default: 0)
--pid=<*file*>	Specify pid file (default: electrad.pid)
--reindex	Rebuild chain state and block index from the blk*.dat files on startup
--reindexaccumulators	
	Reindex the accumulator database on startup
--reindexmoneysupply	Reindex the ECA and zECA money supply statistics on startup
--resync	Delete blockchain folders and resync from scratch on startup
--sysperms	Create new files with system default permissions, instead of umask 077 (only effective with disabled wallet functionality)
--txindex	Maintain a full transaction index, used by the getrawtransaction rpc call (default: 1)
--forcestart	Attempt to force blockchain corruption recovery on startup

Connection options

--addnode=<*ip*>	Add a node to connect to and attempt to keep the connection open
--banscore=<*n*>	Threshold for disconnecting misbehaving peers (default: 100)
--bantime=<*n*>	Number of seconds to keep misbehaving peers from reconnecting (default: 86400)
--bind=<*addr*>	Bind to given address and always listen on it. Use [host]:port notation for IPv6
--connect=<*ip*>	Connect only to the specified node(s)
--discover	Discover own IP addresses (default: 1 when listening and no -externalip or -proxy)

--dns	Allow DNS lookups for -addnode, -seednode and -connect (default: 1)
--dnsseed	Query for peer addresses via DNS lookup, if low on addresses (default: 1 unless -connect)
--externalip=<*ip*>	
	Specify your own public address
--listen	Accept connections from outside (default: 1 if no -proxy or -connect)
--listenonion	Automatically create Tor hidden service (default: 1)
--maxconnections=<*n*>	
	Maintain at most <n> connections to peers (temporary service connections excluded) (default: 125)
--maxreceivebuffer=<*n*>	
	Maximum per-connection receive buffer, <n>*1000 bytes (default: 5000)
--maxsendbuffer=<*n*>	
	Maximum per-connection send buffer, <n>*1000 bytes (default: 1000)
--onion=<ip:port>	
	Use separate SOCKS5 proxy to reach peers via Tor hidden services (default: -proxy)
--onlynet=<*net*>	
	Only connect to nodes in network <net> (ipv4, ipv6 or onion)
---permitbaremultisig	
	Relay non-P2SH multisig (default: 1)
--port=<*port*>	Listen for connections on <port> (default: 9999 or testnet: 19999)
--proxy=<*ip:port*>	
	Connect through SOCKS5 proxy
--seednode=<*ip*>	
	Connect to a node to retrieve peer addresses, and disconnect
--proxyrandomize	

	Randomize credentials for every proxy connection. This enables Tor stream isolation (default: 1)
--seednode=<ip>	
	Connect to a node to retrieve peer addresses, and disconnect
--timeout=<*n*>	Specify connection timeout in milliseconds (minimum: 1, default: 5000)
--torcontrol=<ip>:<port>	
	Tor control port to use if onion listening enabled (default: 127.0.0.1:9051)
--torpassword=<pass>	
	Tor control port password (default: empty)
--upnp	Use UPnP to map the listening port (default: 0)
--whitebind=<addr>	Bind to given address and whitelist peers connecting to it. Use [host]:port notation for IPv6
--whitelist=<netmask>	
	Whitelist peers connecting from the given netmask or IP address. Can be specified multiple times. Whitelisted peers cannot be DoS banned and heir transactions are always relayed, even if they are already in the mempool, useful e.g. for a gateway

Wallet options

--backuppath=<dir|file>	
	Specify custom backup path to add a copy of any wallet backup. If set as dir, every backup generates a timestamped file. If set as file, will rewrite to that file every backup
--createwalletbackups=<n>	
	Number of automatic wallet backups (default: 10)
--custombackupthreshold=<n>	
	Number of custom location backups to retain (default: 1)
--disablewallet	

	Do not load the wallet and disable wallet RPC calls
--keypool=<*n*>	Set key pool size to <n> (default: 1000)
--paytxfee=<*amt*>	
	Fee (in ELECTRA/kB) to add to transactions you send (default: 0.00)
--rescan	Rescan the block chain for missing wallet transactions on startup
--salvagewallet	
	Attempt to recover private keys from a corrupt wallet.dat on startup
--sendfreetransactions	
	Send transactions as zero-fee transactions if possible (default: 0)
--spendzeroconfchange	
	Spend unconfirmed change when sending transactions (default: 1)
--disablesystemnotifications	
	Disable OS notifications for incoming transactions (default: 0)
--txconfirmtarget=<n>	
	If paytxfee is not set, include enough fee so transactions begin confirmation on average within n blocks (default: 1)
--maxtxfee=<amt>	
	Maximum total fees to use in a single wallet transaction, setting too low may abort large transactions (default: 1.00)
--upgradewallet	
	Upgrade wallet to latest format on startup
--wallet=<*file*>	
	Specify wallet file (within data directory) (default: wallet.dat)
--walletnotify=<*cmd*>	
	Execute command when a wallet transaction changes (%s in cmd is replaced by TxID)
-zapwallettxes=<mode>	
	Delete all wallet transactions and only recover those parts of the blockchain through -rescan on startup (1 = keep tx meta data

	ccount owner and payment request information, 2 = drop tx meta data)

Debugging/Testing options

--debug=<*category*>	
	Output debugging information (default: 0, supplying <category> is optional). If <category> is not supplied, output all debugging information.<category> can be: addrman, alert, bench, coindb, db, lock,rand, rpc, selectcoins, tor, mempool, net, proxy, http, libevent, electra, (obfuscation, swiftx, masternode, mnpayments, mnbudget, zero).
--help-debug	Show all debugging options (usage: –help -help-debug)
--gen	Generate coins (default: 0)
--genproclimit=<n>	
	Set the number of threads for coin generation if enabled (-1 = all cores, default: 1)
--logtimestamps	
	Prepend debug output with timestamp (default: 1)
--logips	
	Include IP addresses in debug output (default: 0)
--minrelaytxfee=<amt>	
	Fees (in ECA/Kb) smaller than this are considered zero fee for relaying (default: 0.0001)
--printtoconsole	
	Send trace/debug info to console instead of debug.log file
--shrinkdebugfile	
	Shrink debug.log file on client startup (default: 1 when no -debug)
--testnet	
	Use the test network
-litemode=<n>	

	Disable all Electra specific functionality (Masternodes, Zerocoin, wiftX, Budgeting) (0-1, default: 0)

Staking options

--staking=<n>	Enable staking functionality (0-1, default: 1)
--ecastake=<n>	Enable or disable staking functionality for ECA inputs (0-1, default: 1)
--zecastake=<n>	Enable or disable staking functionality for zECA inputs (0-1, default: 1)
--reservebalance=<amt>	
	Keep the specified amount available for spending at all times (default: 0)

Block creation options

--blockminsize=<*n*>	
	Set minimum block size in bytes (default: 0)
--blockmaxsize=<*n*>	
	Set maximum block size in bytes (default: 750000)
--blockprioritysize=<*n*>	
	Set maximum size of high-priority/low-fee transactions in bytes (default: 10000)

RPC server options

--server	
	Accept command line and JSON-RPC commands
--rest	
	Accept public REST requests (default: 0)
--rpcbind=<addr>	
	Bind to given address to listen for JSON-RPC connections. Use

	[host]:port notation for IPv6. This option can be specified multiple times (default: bind to all interfaces)
--rpccookiefile=<loc>	
	Location of the auth cookie (default: data dir)
--rpcuser=<*user*>	
	Username for JSON-RPC connections
--rpcpassword=<*pw*>	
	Password for JSON-RPC connections
--rpcport=<*port*>	
	Listen for JSON-RPC connections on <port> (default: 9998 or testnet: 19998)
--rpcallowip=<*ip*>	
	Allow JSON-RPC connections from specified source. Valid for <ip> are a single IP (e.g. 1.2.3.4), a network/netmask (e.g. 1.2.3.4/255.255.255.0) or a network/CIDR (e.g. 1.2.3.4/24). This option can be specified multiple times
--rpcthreads=<n>	
	Set the number of threads to service RPC calls (default: 4)

Electra-qt

Electra QT GUI, use same command line options as Electrad with additional options for UI as described below.

Usage

Electra-qt [command-line options]

Start Electra QT GUI

Debugging/Testing options

--debug=<*category*>

	Output debugging information (default: 0, supplying <category> is optional). If <category> is not supplied or if <category> = 1, output all debugging information.<category> can be: addrman, alert, bench, coindb, db, http, libevent, lock, mempool, mempoolrej, net, proxy, prune, rand, reindex, rpc, selectcoins, tor, zmq, electra (or specifically: gobject, instantsend, keepass, masternode, mnpayments, mnsync, privatesend, spork), qt.

UI options

--choosedatadir	
	Choose data directory on startup (default: 0)
--lang=<*lang*>	Set language, for example “de_DE” (default: system locale)
--min	Start minimized
--rootcertificates=<*file*>	
	Set SSL root certificates for payment request (default: -system-)
--splash	Show splash screen on startup (default: 1)

RPC commands

This documentation lists all available RPC commands as of Electra version 2.0.0, and limited documentation on what each command does. For full documentation of arguments, results and examples, type help (“command”) to view full details at the console. You can enter commands either from **Tools > Debug** console in the QT wallet, or using *Electra-cli*

Blockchain

findserial "serial"

Searches the zerocoin database for a zerocoin spend transaction that contains the sp
ecified serial

Arguments:
1. serial (string, required) the serial of a zerocoin spend to search for.

getaccumulatorvalues "height"

Returns the accumulator values associated with a block height

Arguments:
1. height (numeric, required) the height of the checkpoint.

getbestblockhash

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

getblock “hash” (verbose)

If verbose is false, returns a string that is serialized, hex-encoded data for block ‘hash’. If verbose is true, returns an Object with information about block <hash>.

Getblockchaininfo

Returns blockchain information

getblockcount

Returns the number of blocks in the longest block chain.

getblockhash index

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

getblockheader "hash" (verbose)

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

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

getchaintips

getdifficulty

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

getfeeinfo blocks

Returns information about the user's fees and trading volume**.**

getmempoolinfo

returns information about the node’s current transaction memory pool.

getrawmempool (verbose)

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

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 included the mem pool

gettxoutsetinfo

Returns information about UTXO's from Chain

verifychain (numblocks)

Verifies each entry in the local block chain database

Control

debug (0 | 1 | addrman | alert | bench | coindb | db | lock | rand | rpc | selectcoins | mempool | mempoolrej | net | proxy | prune | http | libevent | electra)

Change debug category on the fly. Specify single category or use comma to specify many.

getinfo

Deprecated. Returns an object containing various state info.

help (“command”)

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

stop

Stop Electra Core server.

Electra

checkbudgets

Initiates a buddget check cycle manually

verifychain (numblocks)

Verifies blockchain database.

createmasternodebroadcast "command" ("alias")

createmasternodekey

decodemasternodebroadcast "hexstring"

getbudgetinfo ("proposal")

Show current masternode budgets with optional filter by proposal name.

getbudgetprojection

Show the projection of which proposals will be paid the next cycle.

getbudgetvotes "proposal-name"

Print vote information for a budget proposal.

getmasternodecount

Get masternode count values.

getmasternodeoutputs

Print all masternode transaction outputs.

getmasternodescores (blocks)

Print list of winning masternode by score.

getmasternodestatus

Print masternode status

getmasternodewinners (blocks "filter")

Print the masternode winners for the last *n* blocks

getnextsuperblock

Print the next super block height

getpoolinfo

Returns anonymous pool-related information

listmasternodeconf ("filter")

Print masternode.conf in JSON format.

listmasternodes ("filter")

Get a ranked list of masternodes. Optional filter by txhash, status, or payment address.

masternode "command"...

Set of commands to execute masternode related actions.

masternodeconnect "address"

Attempts to connect to specified masternode address.

masternodecurrent

Get current masternode winner.

masternodedebug

Print masternode status.

mnbudget "command"... ("passphrase")

Vote or show current budgets

mnbudgetrawvote "masternode-tx-hash" masternode-tx-index "proposal-hash" yes|no time "vote-sig"

Compile and relay a proposal vote with provided external signature instead of signing vote internally.

mnbudgetvote "local|many|alias" "votehash" "yes|no" ("alias")

Vote on a budget proposal.

mnfinalbudget "command"... ("passphrase")

Vote or show current budgets.

mnsync "status|reset"

Returns the sync status or resets sync.

preparebudget "proposal-name" "url" payment-count block-start "electra-address" monthy-payment

Prepare proposal for network by signing and creating tx.

relaymasternodebroadcast "hexstring"

spork "name" (value)

Print raw value or active status of sporks.

startmasternode "local|all|many|missing|disabled|alias" lockwallet ("alias")

Attempts to start one or more masternode(s).

submitbudget "proposal-name" "url" payment-count block-start "electra-address" monthy-payment "fee-tx"

Submit proposal to the network.

Generating

getgenerate

PoW Only: Return if the server is set to generate coins or not. The default is false. It is set with the command line argument -gen (or electra.conf setting gen) It can also be set with the setgenerate call.

gethashespersec
setgenerate generate (genproclimit)

PoW Only Set 'generate' true or false to turn generation on or off. Generation is limited to 'genproclimit' processors, -1 is unlimited. See the getgenerate call for the current setting.

Mining

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.

getmininginfo

Returns a json object containing mining-related information.

getnetworkhashps (blocks height)

PoW OnlyReturns the estimated network hashes per second based on the last n blocks.

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

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

reservebalance (reserve amount)

Show or set the reserve amount not participating in network protection. If no parameters provided current setting is printed.

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.

Network

addnode "node" "add|remove|onetry"

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

clearbanned

Clear all banned IPs.

disconnectnode "node"

Immediately disconnects from the specified node.

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.

getconnectioncount

Returns the number of connections to other nodes.

getnettotals

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

getnetworkinfo

Returns an object containing various state info regarding P2P networking.

getpeerinfo

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

listbanned

List all banned IPs/Subnets.

ping

Requests that a ping be sent to all other nodes, to measure ping time.

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

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

Rawtransactions

createrawtransaction [{“txid”:”id”,”vout”:n},…] {“address”:amount,”data”:”hex”,…} (locktime)

Create a transaction spending the given inputs and creating new outputs. Outputs can be addresses or data. 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.

decoderawtransaction “hexstring”

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

decodescript “hex”

Decode a hex-encoded script.

getrawtransaction “txid” (verbose)

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’.

sendrawtransaction “hexstring” (allowhighfees instantsend)

Submits raw transaction (serialized, hex-encoded) to local node and network. Also see createrawtransaction and signrawtransaction calls.

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.

Util

estimatefee nblocks

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

estimatepriority nblocks

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

validateaddress “Electraaddress”

Return information about the given electra address.

verifymessage “electraaddress” “signature” “message”

Verify a signed message.

Wallet

autocombinerewards enable (threshold)

Wallet will automatically monitor for any coins with value below the threshold amount, and combine them if they reside with the same Electra address.

backupwallet "destination"

Safely copies wallet.dat to destination, which can be a directory or a path with filename.

bip38decrypt "electraaddress" "passphrase"

Decrypts and then imports password protected private key.

bip38encrypt "electraaddress" "passphrase"

Encrypts a private key corresponding to 'pivxaddress'.

dumpprivkey “electraaddress”

Reveals the private key corresponding to ‘electraaddress’. Then the importprivkey can be used with this output

dumpwallet “filename”

Dumps all wallet keys in a human-readable format.

encryptwallet “passphrase”

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.

getaccount “electraaddress”

DEPRECATED. Returns the account associated with the given address.

getaccountaddress “account”

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

getaddressesbyaccount “account”

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

getbalance (“account” minconf addlockconf includeWatchonly)

If account is not specified, returns the server’s total available balance. If account is specified (DEPRECATED), returns the balance in the account. Note that the account “” is not the same as leaving the parameter out. The server total may be different to the balance in the default “” account.

getnewaddress (“account”)

Returns a new Electra address for receiving payments. If ‘account’ is specified (DEPRECATED), it is added to the address book so payments received with the address will be credited to ‘account’.

getrawchangeaddress

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

getreceivedbyaccount “account” (minconf addlockconf)

DEPRECATED. Returns the total amount received by addresses with <account> in transactions with specified minimum number of confirmations.

getreceivedbyaddress “electraaddress” (minconf addlockconf)

Returns the total amount received by the given electraaddress in transactions with specified minimum number of confirmations.

getstakesplitthreshold

Returns the threshold for stake splitting.

getstakingstatus

Returns an object containing various staking information.

gettransaction “txid” (includeWatchonly)

Get detailed information about in-wallet transaction <txid>

Getunconfirmedbalance

Returns the server's total unconfirmed balance .

Getwalletinfo

Returns an object containing various wallet state info.

Importaddress

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

importprivkey “electraprivkey” (“label” rescan)

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

importwallet “filename”

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

keypoolrefill (newsize)

Fills the keypool.

listaccounts (minconf addlockconf includeWatchonly)

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

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.

listreceivedbyaccount (minconf addlockconf includeempty includeWatchonly)

DEPRECATED. List balances by account.

listreceivedbyaddress (minconf addlockconf includeempty includeWatchonly)

List balances by receiving address.

listsinceblock (“blockhash” target-confirmations includeWatchonly)

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

listtransactions (“account” count from includeWatchonly)

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

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.

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

move “fromaccount” “toaccount” amount (minconf “comment”)

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

multisend <command>

sendfrom “fromaccount” “toElectraaddress” amount (minconf addlockconf “comment” “comment-to”)

DEPRECATED (use sendtoaddress). Sent an amount from an account to a Electra address.

sendmany “fromaccount” {“address”:amount,…} (minconf addlockconf “comment” [“address”,…] subtractfeefromamount use_is use_ps)

Send multiple times. Amounts are double-precision floating point numbers.

sendtoaddress “Electraaddress” amount (“comment” “comment-to” subtractfeefromamount use_is use_ps)

Send an amount to a given address.

setaccount “Electraaddress” “account”

DEPRECATED. Sets the account associated with the given address.

setstakesplitthreshold value

This will set the output size of your stakes to never be below the given value.

settxfee amount

Set the transaction fee per kB. Overwrites the paytxfee parameter.

signmessage “Electraaddress” “message”

Sign a message with the private key of an address.

walletlock

Removes the wallet encryption key from memory, locking the wallet. After calling this method, you will need to call walletpassphrase again before being able to call any methods which require the wallet to be unlocked.

walletpassphrase “passphrase” timeout (mixingonly)

Stores the wallet decryption key in memory for ‘timeout’ seconds. This is needed prior to performing transactions related to private keys such as sending electras

walletpassphrasechange “oldpassphrase” “newpassphrase”

Changes the wallet passphrase from ‘oldpassphrase’ to ‘newpassphrase’.

Electra-tx

Usage:

electra-tx [options] <hex-tx> [commands] Update hex-encoded electra transaction

electra-tx [options] -create [commands] Create hex-encoded electra transaction

Options:

-?

This help message

-create

Create new, empty TX.

-json

Select JSON output

-txid

Output only the hex-encoded transaction id of the resultant transaction.

-regtest

Enter regression test mode, which uses a special chain in which blocks

can be solved instantly.

-testnet

Use the test network

Commands:

delin=N

Delete input N from TX

delout=N

Delete output N from TX

in=TXID:VOUT

Add input to TX

locktime=N

Set TX lock time to N

nversion=N

Set TX version to N

outaddr=VALUE:ADDRESS

Add address-based output to TX

outscript=VALUE:SCRIPT

Add raw script output to TX

sign=SIGHASH-FLAGS

Add zero or more signatures to transaction. This command requires JSON registers:prevtxs=JSON object, privatekeys=JSON object. See signrawtransaction docs for format of sighash flags, JSON objects.

Register Commands:

load=NAME:FILENAME

Load JSON file FILENAME into register NAME

set=NAME:JSON-STRING

Set register NAME to given JSON-STRING