Litecoin Utilities Documentation

Release 0.4.9

Konstantinos Karasavvas

CONTENTS

1	Keys and Addresses module	3
2	Transactions module	9
3	Script module	15
4	Proxy module	17
5	Indices and tables	19
Рy	Python Module Index	21
Index		23

Contents:

CONTENTS 1

2 CONTENTS

KEYS AND ADDRESSES MODULE

class keys.Address(address=None, hash160=None, script=None)

Represents a Bitcoin address

hash160

the hash160 string representation of the address; hash160 represents two consequtive hashes of the public key or the redeam script, first a SHA-256 and then an RIPEMD-160

Type str

from_address (address)

instantiates an object from address string encoding

from_hash160 (hash160_str)

instantiates an object from a hash160 hex string

from_script (redeem_script)

instantiates an object from a redeem_script

to_string()

returns the address's string encoding

to_hash160()

returns the address's hash160 hex string representation

Raises

- TypeError No parameters passed
- ValueError If an invalid address or hash160 is provided.

classmethod from_address(address)

Creates and address object from an address string

classmethod from_hash160(hash160)

Creates and address object from a hash160 string

classmethod from_script(script)

Creates and address object from a Script object

to_hash160()

Returns as hash160 hex string

to_string()

Returns as address string

Pseudocode:

```
network_prefix = (1 byte version number)
                data = network_prefix + hash160_bytes
                data_hash = SHA-256( SHA-256( hash160_bytes ) )
                checksum = (first 4 bytes of data_hash)
                address_bytes = Base58CheckEncode( data + checksum )
class keys.P2pkhAddress(address=None, hash160=None)
     Encapsulates a P2PKH address.
     Check Address class for details
     to_script_pub_key()
          returns the scriptPubKey (P2PKH) that corresponds to this address
     get_type()
          returns the type of address
     get_type()
          Returns the type of address
     to_script_pub_key()
          Returns the scriptPubKey (P2PKH) that corresponds to this address
class keys.P2shAddress(address=None, hash160=None, script=None)
     Encapsulates a P2SH address.
     Check Address class for details
     get_type()
          returns the type of address
     get_type()
          Returns the type of address
class keys. P2wpkhAddress (address=None, witness_hash=None, version='p2wpkhv0')
     Encapsulates a P2WPKH address.
     Check Address class for details
     to_script_pub_key()
          returns the scriptPubKey of a P2WPKH witness script
     get_type()
          returns the type of address
     get_type()
          Returns the type of address
     to_script_pub_key()
          Returns the scriptPubKey of a P2WPKH witness script
class keys. P2wshAddress (address=None, witness hash=None, script=None, version='p2wshv0')
     Encapsulates a P2WSH address.
     Check Address class for details
     from_script (witness_script)
          instantiates an object from a witness_script
     get_type()
          returns the type of address
```

```
get_type()
          Returns the type of address
     to_script_pub_key()
          Returns the scriptPubKey of a P2WPKH witness script
class keys.PrivateKey(wif=None, secret exponent=None)
     Represents an ECDSA private key.
     key
          the raw key of 32 bytes
               Type bytes
     from wif(wif)
          creates an object from a WIF of WIFC format (string)
     to_wif(compressed=True)
          returns as WIFC (compressed) or WIF format (string)
     to_bytes()
          returns the key's raw bytes
     sign message (message, compressed=True)
          signs the message's digest and returns the signature
     sign_transaction (tx, compressed=True)
          signs the transaction's digest and returns the signature
     get_public_key()
          returns the corresponding PublicKey object
     classmethod from_wif(wif)
          Creates key from WIFC or WIF format key
     get_public_key()
          Returns the corresponding PublicKey
     sign_message (message, compressed=True)
          Signs the message with the private key (deterministically)
          Bitcoin uses a compact format for message signatures (for tx sigs it uses normal DER format). The format
          has the normal r and s parameters that ECDSA signatures have but also includes a prefix which encodes
          extra information. Using the prefix the public key can be reconstructed when verifying the signature.
          Prefix values:
                27 - 0x1B = first key with even y
                28 - 0x1C = first key with odd y
                29 - 0x1D = second key with even y
                30 - 0x1E = second key with odd y
          If key is compressed add 4 (31 - 0x1F, 32 - 0x20, 33 - 0x21, 34 - 0x22 respectively)
          Returns a Bitcoin compact signature in Base64
     to_bytes()
          Returns key's bytes
     to wif(compressed=True)
          Returns key in WIFC or WIF string
```

```
Pseudocode:
                network_prefix = (1 byte version number)
                data = network_prefix + (32 \text{ bytes number/key}) [ + 0x01 \text{ if compressed} ]
                data_hash = SHA-256(SHA-256(data))
                checksum = (first 4 bytes of data_hash)
                wif = Base58CheckEncode( data + checksum )
class keys.PublicKey(hex_str)
     Represents an ECDSA public key.
     key
          the raw public key of 64 bytes (x, y coordinates of the ECDSA curve)
               Type bytes
     from hex (hex str)
          creates an object from a hex string in SEC format
     from_message_signature(signature)
          NO-OP!
     verify_message (address, signature, message)
          Class method that constructs the public key, confirms the address and verifies the signature
     verify (signature, message)
          returns true if the message was signed with this public key's corresponding private key.
     to hex(compressed=True)
          returns the key as hex string (in SEC format - compressed by default)
     to_bytes()
          returns the key's raw bytes
     to_hash160()
          returns the hash160 hex string of the public key
     get_address (compressed=True))
          returns the corresponding P2pkhAddress object
     get_segwit_address()
          returns the corresponding P2wpkhAddress object
     classmethod from hex(hex str)
          Creates a public key from a hex string (SEC format)
     get_address(compressed=True)
          Returns the corresponding P2PKH Address (default compressed)
     get_segwit_address()
          Returns the corresponding P2WPKH address
          Only compressed is allowed. It is otherwise identical to normal P2PKH address.
     to_bytes()
          Returns key's bytes
     to hash160 (compressed=True)
          Returns the RIPEMD(SHA256()) of the public key in hex
     to hex(compressed=True)
          Returns public key as a hex string (SEC format - compressed by default)
```

verify (signature, message)

Verifies that the message was signed with this public key's corresponding private key.

classmethod verify_message(address, signature, message)

Creates a public key from a message signature and verifies message

Bitcoin uses a compact format for message signatures (for tx sigs it uses normal DER format). The format has the normal r and s parameters that ECDSA signatures have but also includes a prefix which encodes extra information. Using the prefix the public key can be reconstructed from the signature.

Prefix values:

```
27 - 0x1B = first key with even y
```

28 - 0x1C =first key with odd y

29 - 0x1D = second key with even y

30 - 0x1E = second key with odd y

If key is compressed add 4(31 - 0x1F, 32 - 0x20, 33 - 0x21, 34 - 0x22 respectively)

Raises ValueError - If signature is invalid

class keys. SegwitAddress (address=None, witness_hash=None, script=None, version='p2wpkhv0')
Represents a Bitcoin segwit address

Note that currently the python bech32 reference implementation is used (by Pieter Wuille).

witness_hash

the hash string representation of either the address; it can be either a public key hash (P2WPKH) or the hash of the script (P2WSH)

Type str

from_address (address)

instantiates an object from address string encoding

from_hash (hash_str)

instantiates an object from a hash hex string

from_script (witness_script)

instantiates an object from a witness_script

to_string()

returns the address's string encoding (Bech32)

$\verb"to_hash"()$

returns the address's hash hex string representation

Raises

- TypeError No parameters passed
- ValueError If an invalid address or hash is provided.

classmethod from_address(address)

Creates and address object from an address string

classmethod from hash(witness hash)

Creates and address object from a hash string

classmethod from_script(script)

Creates and address object from a Script object

to_hash()

Returns as hash hex string

to_string()

Returns as address string

Uses a segwit's python reference implementation for now. (TODO)

TRANSACTIONS MODULE

class transactions.Locktime (value)

Helps setting up appropriate locktime.

value

The value of the block height or the Unix epoch (seconds from 1 Jan 1970 UTC)

Type int

for transaction()

Serializes the locktime as required in a transaction

Raises ValueError – if the value is not within range of 2 bytes.

for_transaction()

Creates a timelock as expected from Transaction

class transactions.Sequence(seq_type, value=None, is_type_block=True)

Helps setting up appropriate sequence. Used to provide the sequence to transaction inputs and to scripts.

value

The value of the block height or the 512 seconds increments

Type int

seq_type

Specifies the type of sequence (TYPE_RELATIVE_TIMELOCK | TYPE_ABSOLUTE_TIMELOCK | TYPE_REPLACE_BY_FEE

Type int

is_type_block

If type is TYPE_RELATIVE_TIMELOCK then this specifies its type (block height or 512 secs increments)

Type bool

for_input_sequence()

Serializes the relative sequence as required in a transaction

for_script()

Returns the appropriate integer for a script; e.g. for relative timelocks

Raises ValueError – if the value is not within range of 2 bytes.

for_input_sequence()

Creates a relative timelock sequence value as expected from TxInput sequence attribute

```
for_script()
          Creates a relative/absolute timelock sequence value as expected in scripts
sion=b^x02\\x00\\x00\\x00', has\_segwit=False, witnesses=[])
     Represents a Bitcoin transaction
     inputs
          A list of all the transaction inputs
              Type list (TxInput)
     outputs
          A list of all the transaction outputs
              Type list (TxOutput)
     locktime
          The transaction's locktime parameter
              Type bytes
     version
          The transaction version
              Type bytes
     has_segwit
          Specifies a tx that includes segwit inputs
              Type bool
     witnesses
          The witness scripts that correspond to the inputs
              Type list (Script)
     stream()
          Converts Transaction to bytes
     serialize()
          Converts Transaction to hex string
     get_txid()
          Calculates txid and returns it
     get hash()
          Calculates tx hash (wtxid) and returns it
     get_wtxid()
          Calculates tx hash (wtxid) and returns it
     get size()
          Calculates the tx size
     get_vsize()
          Calculates the tx segwit size
     copy()
          creates a copy of the object (classmethod)
     get_transaction_digest (txin_index, script, sighash)
          returns the transaction input's digest that is to be signed according
     get_transaction_segwit_digest (txin_index, script, amount, sighash)
          returns the transaction input's segwit digest that is to be signed according to sighash
```

```
classmethod copy (tx)
     Deep copy of Transaction
get hash()
     Hashes the serialized (bytes) tx including segwit marker and witnesses
get size()
     Gets the size of the transaction
get transaction digest (txin index, script, sighash=1)
     Returns the transaction's digest for signing.
     SIGHASH types (see constants.py):
          SIGHASH_ALL - signs all inputs and outputs (default)
          SIGHASH_NONE - signs all of the inputs
          SIGHASH_SINGLE - signs all inputs but only txin_index output
          SIGHASH ANYONECANPAY (only combined with one of the above)
          - with ALL - signs all outputs but only txin_index input
          - with NONE - signs only the txin_index input
          - with SINGLE - signs txin_index input and output
     txin index
         The index of the input that we wish to sign
             Type int
     script
         The scriptPubKey of the UTXO that we want to spend
             Type list (string)
     sighash
         The type of the signature hash to be created
             Type int
get_transaction_segwit_digest (txin_index, script, amount, sighash=1)
     Returns the segwit transaction's digest for signing.
     SIGHASH types (see constants.py):
          SIGHASH ALL - signs all inputs and outputs (default)
          SIGHASH_NONE - signs all of the inputs
          SIGHASH_SINGLE - signs all inputs but only txin_index output
          SIGHASH_ANYONECANPAY (only combined with one of the above)
          - with ALL - signs all outputs but only txin_index input
          - with NONE - signs only the txin index input
          - with SINGLE - signs txin index input and output
     txin index
         The index of the input that we wish to sign
             Type int
     script
         The scriptPubKey of the UTXO that we want to spend
```

```
Type list (string)
           amount
               The amount of the UTXO to spend is included in the signature for segwit (in satoshis)
                    Type int/float/Decimal
           sighash
               The type of the signature hash to be created
                    Type int
      get_txid()
           Hashes the serialized (bytes) tx to get a unique id
      get_vsize()
           Gets the virtual size of the transaction.
           For non-segwit txs this is identical to get_size(). For segwit txs the marker and witnesses length needs to
           be reduced to 1/4 of its original length. Thus it is substructed from size and then it is divided by 4 before
           added back to size to produce vsize (always rounded up).
           https://en.litecoin.it/wiki/Weight_units
      get_wtxid()
           Hashes the serialized (bytes) tx including segwit marker and witnesses
      serialize()
           Converts to hex string
      stream(has_segwit)
           Converts to bytes
class transactions. TxInput (txid, txout_index, script_sig=<litecoinutils.script.Script object>, se-
                                       quence=b xff xff xff xff xff 
      Represents a transaction input.
      A transaction input requires a transaction id of a UTXO and the index of that UTXO.
      txid
           the transaction id as a hex string (little-endian as displayed by tools)
               Type str
      txout_index
           the index of the UTXO that we want to spend
               Type int
      script_sig
           the op code and data of the script as string
               Type list (strings)
      sequence
           the input sequence (for timelocks, RBF, etc.)
               Type bytes
      stream()
           converts TxInput to bytes
           creates a copy of the object (classmethod)
      classmethod copy (txin)
           Deep copy of TxInput
```

```
stream()
          Converts to bytes
class transactions.TxOutput (amount, script_pubkey)
     Represents a transaction output
     amount
          the value we want to send to this output in satoshis
              Type int/float/Decimal
     script_pubkey
          the script that will lock this amount
              Type list (string)
     stream()
          converts TxInput to bytes
     copy()
          creates a copy of the object (classmethod)
     classmethod copy(txout)
          Deep copy of TxOutput
     stream()
          Converts to bytes
```

THREE

SCRIPT MODULE

class script.Script(script)

Represents any script in Bitcoin

A Script contains just a list of OP_CODES and also knows how to serialize into bytes

script

the list with all the script OP CODES and data

Type list

to_bytes()

returns a serialized byte version of the script

get_script()

returns the list of strings that makes up this script

Raises ValueError - If string data is too large or integer is negative

classmethod copy (script)

Deep copy of Script

get_script()

Returns script as array of strings

to_bytes (segwit=False)

Converts the script to bytes

If an OP code the appropriate byte is included according to: https://en.litecoin.it/wiki/Script If not consider it data (signature, public key, public key hash, etc.) and and include with appropriate OP_PUSHDATA OP code plus length

to_hex()

Converts the script to hexadecimal

to_p2sh_script_pub_key()

Converts script to p2sh scriptPubKey (locking script)

Calculates the hash160 (via the address) of the script and uses it to construct a P2SH script.

to_p2wsh_script_pub_key()

Converts script to p2wsh scriptPubKey (locking script)

Calculates the sha256 of the script and uses it to construct a P2WSH script.

CHAPTER

FOUR

PROXY MODULE

 $\begin{tabular}{ll} \textbf{class} & \texttt{proxy}. \textbf{NodeProxy} (\textit{rpcuser=None}, \textit{rpcpassword=None}, \textit{host=None}, \textit{port=None}) \\ & \textbf{Simple Bitcoin node proxy that can call all of Bitcoin's JSON-RPC functionality}. \\ \end{tabular}$

proxy

a litecoinrpc AuthServiceProxy object

Type object

get_proxy()

Returns litecoinrpc AuthServiceProxy object

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

```
k
keys, 3
p
proxy, 17
s
script, 15
t
transactions, 9
```

22 Python Module Index

INDEX

A	G		
Address (class in keys), 3	<pre>get_address() (keys.PublicKey method), 6</pre>		
amount (transactions.Transaction attribute), 12	get_hash() (transactions.Transaction method), 10, 11		
amount (transactions.TxOutput attribute), 13	<pre>get_proxy() (proxy.NodeProxy method), 17</pre>		
_	<pre>get_public_key() (keys.PrivateKey method), 5</pre>		
C	<pre>get_script() (script.Script method), 15</pre>		
copy () (script.Script class method), 15	<pre>get_segwit_address() (keys.PublicKey method),</pre>		
copy () (transactions. Transaction class method), 11	6		
copy () (transactions. Transaction method), 10	<pre>get_size() (transactions.Transaction method), 10, 11</pre>		
copy () (transactions.TxInput class method), 12	<pre>get_transaction_digest() (transac-</pre>		
copy () (transactions.TxInput method), 12	tions.Transaction method), 10, 11		
copy () (transactions.TxOutput class method), 13	<pre>get_transaction_segwit_digest() (transac-</pre>		
copy () (transactions.TxOutput method), 13	tions.Transaction method), 10, 11		
_	<pre>get_txid() (transactions.Transaction method), 10, 12</pre>		
F	<pre>get_type() (keys.P2pkhAddress method), 4</pre>		
<pre>for_input_sequence() (transactions.Sequence</pre>	<pre>get_type() (keys.P2shAddress method), 4</pre>		
method), 9	<pre>get_type() (keys.P2wpkhAddress method), 4</pre>		
<pre>for_script() (transactions.Sequence method), 9</pre>	get_type() (keys.P2wshAddress method), 4		
<pre>for_transaction() (transactions.Locktime</pre>	get_vsize() (transactions.Transaction method), 10,		
method), 9	12		
<pre>from_address() (keys.Address class method), 3</pre>	<pre>get_wtxid() (transactions.Transaction method), 10,</pre>		
<pre>from_address() (keys.Address method), 3</pre>	12		
<pre>from_address() (keys.SegwitAddress class method),</pre>	Н		
7	••		
<pre>from_address() (keys.SegwitAddress method), 7</pre>	has_segwit (transactions.Transaction attribute), 10		
<pre>from_hash() (keys.SegwitAddress class method), 7</pre>	hash160 (keys.Address attribute), 3		
<pre>from_hash() (keys.SegwitAddress method), 7</pre>	1		
from_hash160() (keys.Address class method), 3			
from_hash160() (keys.Address method), 3	inputs (transactions.Transaction attribute), 10		
from_hex() (keys.PublicKey class method), 6	is_type_block (transactions.Sequence attribute), 9		
from_hex() (keys.PublicKey method), 6	K		
from_message_signature() (keys.PublicKey			
method), 6	key (keys.PrivateKey attribute), 5		
from_script() (keys.Address class method), 3	key (keys.PublicKey attribute), 6		
from_script() (keys.Address method), 3	keys		
from_script() (keys.P2wshAddress method), 4	module, 3		
from_script() (keys.SegwitAddress class method), 7	1		
from_script() (keys.SegwitAddress method), 7			
from_wif() (keys.PrivateKey class method), 5	Locktime (class in transactions), 9		
<pre>from_wif() (keys.PrivateKey method), 5</pre>	locktime (transactions.Transaction attribute), 10		

M	to_hex() (script.Script method),	15	
module	to_p2sh_script_pub_key()	(script.Script	
keys, 3	method), 15		
proxy, 17	to_p2wsh_script_pub_key(() (script.Script	
script,15	method), 15		
transactions,9	to_script_pub_key() method), 4	(keys.P2pkhAddress	
N	to_script_pub_key()	(keys.P2wpkhAddress	
NodeProxy (class in proxy), 17	method), 4		
0	<pre>to_script_pub_key() method), 5</pre>	(keys.P2wshAddress	
outputs (transactions.Transaction attribute), 10	to_string() (keys.Address meth	hod), 3	
outputs (nansactions. Transaction autibute), 10	to_string() (keys.SegwitAddress method), 7, 8		
P	to_wif() (keys.PrivateKey metho		
P2pkhAddress (class in keys), 4	Transaction (class in transaction	ons), 10	
P2shAddress (class in keys), 4	transactions		
P2wpkhAddress (class in keys), 4	module, 9	() 10	
P2wshAddress (class in keys), 4	txid (transactions.TxInput attribu		
PrivateKey (class in keys), 5	txin_index (transactions.Transa TxInput (class in transactions), 1		
proxy	txout_index (transactions.TxIn		
module, 17	TxOutput (class in transactions),		
proxy (proxy.NodeProxy attribute), 17	1xouepue (etass in transactions),	13	
PublicKey (class in keys), 6	V		
S	value (transactions.Locktime attravalue (transactions.Sequence attr		
script	verify() (keys.PublicKey method		
module, 15	verify_message() (keys.Publi		
Script (class in script), 15	verify_message() (keys.Publi		
script (script.Script attribute), 15	version (transactions.Transactio		
script (transactions.Transaction attribute), 11		,, 10	
<pre>script_pubkey (transactions.TxOutput attribute), 13</pre>	W		
script_sig (transactions.TxInput attribute), 12	witness_hash(keys.SegwitAddr	ress attribute). 7	
SegwitAddress (class in keys), 7	witnesses (transactions. Transac		
seq_type (transactions.Sequence attribute), 9		,,,	
Sequence (class in transactions), 9			
sequence (transactions.TxInput attribute), 12			
serialize() (transactions.Transaction method), 10, 12			
sighash (transactions.Transaction attribute), 11, 12			
<pre>sign_message() (keys.PrivateKey method), 5</pre>			
<pre>sign_transaction() (keys.PrivateKey method), 5</pre>			
stream() (transactions.Transaction method), 10, 12			
stream() (transactions.TxInput method), 12			
stream() (transactions.TxOutput method), 13			
Т			
to_bytes() (keys.PrivateKey method), 5			
to_bytes() (keys.PublicKey method), 6			
to_bytes() (script.Script method), 15			
to_hash() (keys.SegwitAddress method), 7, 8			
to_hash160() (keys.Address method), 3			
to_hash160() (keys.PublicKey method), 6			
to_hex() (keys.PublicKey method), 6			

24 Index