

---

# **Bitcoin Utilities Documentation**

***Release 0.4.11***

**Konstantinos Karasavvas**

**Nov 04, 2020**



# CONTENTS

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



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 consecutive hashes of the public key or the redeem 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\_input** (*tx, txin\_index, script, sighash=SIGHASH\_ALL*)  
signs the transaction's digest for a particular index and returns the signature.

**sign\_segwit\_input** (*tx, txin\_index, script, amount, sighash=SIGHASH\_ALL*)  
signs the transaction's digest for a particular index and amount 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 bytes number/key) [ + 0x01 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)

**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

**class** transactions.**Transaction**(inputs=None, outputs=None, locktime=b'x00x00x00x00', version=b'x02x00x00x00', has\_segwit=False, witnesses=None)

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 substructured from size and then it is divided by 4 before added back to size to produce vsize (always rounded up).

[https://en.bitcoin.it/wiki/Weight\\_units](https://en.bitcoin.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*=<bitcoinutils.script.Script object>, *sequence*=b'\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

**copy()**

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



## 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.bitcoin.it/wiki/Script> If not consider it data (signature, public key, public key hash, etc.) 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.



## PROXY MODULE

**class** proxy.**NodeProxy** (*rpcuser=None, rpcpassword=None, host=None, port=None*)

Simple Bitcoin node proxy that can call all of Bitcoin's JSON-RPC functionality.

**proxy**

a bitcoinrpc AuthServiceProxy object

**Type** object

**get\_proxy** ()

Returns bitcoinrpc AuthServiceProxy object



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`





## PYTHON MODULE INDEX

### k

keys, [3](#)

### p

proxy, [17](#)

### s

script, [15](#)

### t

transactions, [9](#)



## A

Address (class in keys), 3  
 amount (transactions.Transaction attribute), 12  
 amount (transactions.TxOutput attribute), 13

## C

copy () (script.Script class method), 15  
 copy () (transactions.Transaction class method), 11  
 copy () (transactions.Transaction method), 10  
 copy () (transactions.TxInput class method), 12  
 copy () (transactions.TxInput method), 12  
 copy () (transactions.TxOutput class method), 13  
 copy () (transactions.TxOutput method), 13

## F

for\_input\_sequence () (transactions.Sequence method), 9  
 for\_script () (transactions.Sequence method), 9  
 for\_transaction () (transactions.Locktime method), 9  
 from\_address () (keys.Address class method), 3  
 from\_address () (keys.Address method), 3  
 from\_address () (keys.SegwitAddress class method), 7  
 from\_address () (keys.SegwitAddress method), 7  
 from\_hash () (keys.SegwitAddress class method), 8  
 from\_hash () (keys.SegwitAddress method), 7  
 from\_hash160 () (keys.Address class method), 3  
 from\_hash160 () (keys.Address method), 3  
 from\_hex () (keys.PublicKey class method), 6  
 from\_hex () (keys.PublicKey method), 6  
 from\_message\_signature () (keys.PublicKey method), 6  
 from\_script () (keys.Address class method), 3  
 from\_script () (keys.Address method), 3  
 from\_script () (keys.P2wshAddress method), 4  
 from\_script () (keys.SegwitAddress class method), 8  
 from\_script () (keys.SegwitAddress method), 7  
 from\_wif () (keys.PrivateKey class method), 5  
 from\_wif () (keys.PrivateKey method), 5

## G

get\_address () (keys.PublicKey method), 6  
 get\_hash () (transactions.Transaction method), 10, 11  
 get\_proxy () (proxy.NodeProxy method), 17  
 get\_public\_key () (keys.PrivateKey method), 5  
 get\_script () (script.Script method), 15  
 get\_segwit\_address () (keys.PublicKey method), 6  
 get\_size () (transactions.Transaction method), 10, 11  
 get\_transaction\_digest () (transactions.Transaction method), 10, 11  
 get\_transaction\_segwit\_digest () (transactions.Transaction method), 10, 11  
 get\_txid () (transactions.Transaction method), 10, 12  
 get\_type () (keys.P2pkhAddress method), 4  
 get\_type () (keys.P2shAddress method), 4  
 get\_type () (keys.P2wpkhAddress method), 4  
 get\_type () (keys.P2wshAddress method), 4  
 get\_vsize () (transactions.Transaction method), 10, 12  
 get\_wtxid () (transactions.Transaction method), 10, 12

## H

has\_segwit (transactions.Transaction attribute), 10  
 hash160 (keys.Address attribute), 3

## I

inputs (transactions.Transaction attribute), 10  
 is\_type\_block (transactions.Sequence attribute), 9

## K

key (keys.PrivateKey attribute), 5  
 key (keys.PublicKey attribute), 6  
 keys (module), 3

## L

Locktime (class in transactions), 9  
 locktime (transactions.Transaction attribute), 10

## N

NodeProxy (class in proxy), 17

## O

`outputs` (*transactions.Transaction* attribute), 10

## P

`P2pkhAddress` (class in *keys*), 4

`P2shAddress` (class in *keys*), 4

`P2wpkhAddress` (class in *keys*), 4

`P2wshAddress` (class in *keys*), 4

`PrivateKey` (class in *keys*), 5

`proxy` (module), 17

`proxy` (*proxy.NodeProxy* attribute), 17

`PublicKey` (class in *keys*), 6

## S

`Script` (class in *script*), 15

`script` (module), 15

`script` (*script.Script* attribute), 15

`script` (*transactions.Transaction* attribute), 11

`script_pubkey` (*transactions.TxOutput* attribute), 13

`script_sig` (*transactions.TxInput* attribute), 12

`SegwitAddress` (class in *keys*), 7

`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

`sign_input()` (*keys.PrivateKey* method), 5

`sign_message()` (*keys.PrivateKey* method), 5

`sign_segwit_input()` (*keys.PrivateKey* method), 5

`stream()` (*transactions.Transaction* method), 10, 12

`stream()` (*transactions.TxInput* method), 12

`stream()` (*transactions.TxOutput* method), 13

## T

`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, 7

`to_hex()` (*script.Script* method), 15

`to_p2sh_script_pub_key()` (*script.Script* method), 15

`to_p2wsh_script_pub_key()` (*script.Script* method), 15

`to_script_pub_key()` (*keys.P2pkhAddress* method), 4

`to_script_pub_key()` (*keys.P2wpkhAddress* method), 4

`to_script_pub_key()` (*keys.P2wshAddress* method), 5

`to_string()` (*keys.Address* method), 3

`to_string()` (*keys.SegwitAddress* method), 7, 8

`to_wif()` (*keys.PrivateKey* method), 5

`Transaction` (class in *transactions*), 10

`transactions` (module), 9

`txid` (*transactions.TxInput* attribute), 12

`txin_index` (*transactions.Transaction* attribute), 11

`TxInput` (class in *transactions*), 12

`txout_index` (*transactions.TxInput* attribute), 12

`TxOutput` (class in *transactions*), 13

## V

`value` (*transactions.Locktime* attribute), 9

`value` (*transactions.Sequence* attribute), 9

`verify()` (*keys.PublicKey* method), 6, 7

`verify_message()` (*keys.PublicKey* class method), 7

`verify_message()` (*keys.PublicKey* method), 6

`version` (*transactions.Transaction* attribute), 10

## W

`witness_hash` (*keys.SegwitAddress* attribute), 7

`witnesses` (*transactions.Transaction* attribute), 10