web3fsnpy

Release 1.0.2

Dec 14, 2020

Contents:

web3fsnpy is a python library for interacting with Fusion. Its API is derived from the Web3.py and Web3.js Javascript API and should be familiar to anyone who has used `web3.py` which it extends. It mirrors Fusion's web3-fusion-extend Javascript library in its functionality.

By creating a pythonic version of the API for it's blockchain, Fusion Foundation has made it possible to easily unlock all the functionality that makes Fusion unique. With only single function calls, a user can now create assets, send tokens, or generate time locks to unlock the time value of assets and other cryptocurrencies locked in to Fusion's blockchain.

Because python is easy to learn and is platform independent, every user now has access to Fusion's features and can combine them with every other python module, including math and scientific modules, specialist financial modules, to assist them in developing feature rich applications.

CHAPTER 1

Installation

These instructions work for Ubuntu (> v18.04) and Debian (> v10 buster). There may be some variations for other distros.

1.1 From a wheel Using pip3

Install some dependencies (you will need > python3.6) :-

```
#> sudo apt install python3 python3-pip
#> sudo pip3 install web3fsnpy  (or pip3 install web3fsnpy --user if you want to␣
↪install a username only copy)
```

1.2 Quick Start

You can find some example python programs at *Code Examples* designed to demonstrate the API's functionality. These will be added to as new functions are developed.

You will probably need to set the environmental variable FSN_PRIVATE_KEY to be able to use any write transaction methods. Get your private key from your Fusion wallet (click on 'View details') and then :-

```
#> export FSN_PRIVATE_KEY=XXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Alternatively you can generate your private key using your wallet JSON file and an input password. The way to do this is illustrated in *Code Examples*

To update (frequent updates available) just type :-

```
#> sudo pip3 install --upgrade web3fsnpy  (or pip3 install --upgrade web3fsnpy --user)
```

Now you need to update the PYTHONPATH environmental variable to your .bashrc file assuming that you are in the folder web3fsnpy :-

```
#> echo "export PYTHONPATH=$PWD:$PWD/web3fsnpy:$PYTHONPATH">>~/.bashrc
```

Now restart your shell to activate the PYTHONPATH.

Note that you may be able to substitute the command pip for pip3 and python for python3, depending on how your system is set up. We assume from now on that pip and python have the version 3.x

1.3 Developer setup

For a developer setup, although it is not required, you will likely need to install Ethereum's web3.py according to the instructions at https://github.com/ethereum/web3.py

To install web3fsnpy :-

```
#> git clone https://github.com/FUSIONFoundation/web3fsnpy.git
```

The dependencies are listed in the file requirements.txt

It is best practice to operate within a virtualenv when modifying code, so as to isolate dependency issues. The –no-site-packages option below prevents usage of any other python modules that may exist on your system, but which might cause an inconsistency with web3fsnpy :-

```
#> virtualenv --no-site-packages -p /usr/bin/python3 env   # assuming that python3 is␣
↪there - check with 'which python3'
#> source env/bin/activate              # this changes the prompt and puts you into␣
↪your virtualenv
```

You should check that you are now using local versions of python and pip :-

```
#> which python     # should output an answer within your env/bin/python folder
#> which pip        # same
#> python --version # should indicate a 3.x version
#> pip --version    # same
```

To install the python dependencies in this virtual environment from the file requirements.txt :-

```
#> pip install -r requirements.txt
```

Check that the scope includes the correct python module versions :-

```
#> pip list  (or pip show <module name> )
```

Sometimes a pre-existing install may have a higher version number of a module, which can cause unpredictable results.

1.4 Connection to the Blockchain

There are three ways to connect to the blockchain:- HTTP, WebSocket and IPC. If you are not running a node on your machine, then WebSocket is preferred over HTTP, since it allows asynchronous bi-directional communication.

There are two 'networks' :- 'mainnet' and 'testnet'.

The 'default' 'gateway' values hardcoded into the Fsn class constructor __init__(linkToChain) method are as follows:-

WebSocket (testnet):- 'wss://testnetpublicgateway1.fusionnetwork.io:10001' WebSocket (mainnet):- 'wss://mainnetpublicgateway1.fusionnetwork.io:10001' HTTP (testnet):- 'https://testnetpublicgateway1.

fusionnetwork.io:10000/' HTTP (mainnet):- 'https://mainnetpublicgateway1.fusionnetwork.io:10000/' IPC :- '/home/root/fusion-node/data/efsn.ipc'

These can all be overridden using strings

1.5 Sanity Check

You can check that your installation has been successful as follows :-

```
>>
#web3fusion
from  web3fsnpy import Fsn

linkToChain = {
    'network'     : 'testnet',                          # One of 'testnet', or
↪'mainnet'
    'provider'    : 'HTTP',                             # One of 'WebSocket', 'HTTP',␣
↪or 'IPC'
    'gateway'     : 'default',                          # Either set to 'default', or␣
↪uri endpoint
    #'private_key'     : os.environ["FSN_PRIVATE_KEY"],  # comment out for just read␣
↪operations
}

#

web3fsn = Fsn(linkToChain)

print('Current block height is ',web3fsn.blockNumber)
```

CHAPTER 2

Functions in the *Fsn* Class

Fsn extends *web3.eth*

There are many functions that can be used via the *Fsn* class that are not described here, but are available from the *web3.eth* base class. These include functions to handle smart contracts. Please consult the web3.py documentation for their description and usage.

The functions in *Fsn* are split up into categories below.

2.1 Tickets

ticketPrice()

2.1.1 ticketPrice

def ticketPrice(self, block_identifier='latest'): """Get the most recent ticket price

Args:

block_idenfifier: blockNo (int), 'latest', 'earliest', or 'pending'

Returns: ticket_price (int) in Wei

"""

```
# Get the ticket price

ticket_price = web3fsn.ticketPrice()
ticket_price = web3fsn.fromWei(ticket_price,'ether')

print('\nThe ticket price = ',ticket_price,' FSN')
```

getStakeInfo()

2.1.2 getStakeInfo

def getStakeInfo(self, block_identifier='latest'): """Get the latest information about the nodes and their tickets

Args:

block_idenfifier: blockNo (int), 'latest', 'earliest', or 'pending'

Returns: stake_info (Attribute dict) 'stakeInfo'[{'owner': pub_key1 (hex str), 'tickets':nTickets1 (int)},{'owner':pub_key2, 'tickets':nTickets2 }...] 'summary' (Attribute dict) {'totalMiners': tot, 'totalTickets':tot_tick}

"""

```
# Get the staking information

stake_info = web3fsn.getStakeInfo()
#print(stake_info)

for node in range(stake_info.summary.totalMiners):
    print(stake_info.stakeInfo[node].owner, ' has ', stake_info.stakeInfo[node].
↪tickets, ' tickets')
```

Output :-

```
>>>
0x76c2ae4281fe1ee1a79ccbdda2516d4d7eb0eb37  has  380  tickets
0x88817ef0545ca562530f9347b20138edecfd8e30  has  374  tickets
0x494a792d704e24309fd778641683502fd30f9913  has  290  tickets
0x37afe6319dbd980741cd3bfe701f196694d20564  has  244  tickets
0xced7849e100c92768bebda4575db63301f5515e2  has  222  tickets
0x47bb222e76ff205677132fba7c6cfcddfb4128d2  has  178  tickets
0xb2a46485d73b47af2c7a62ed1868c48a557dddc0  has  136  tickets
0x1a77c95b429c0c5646476487d3faab422b541b18  has  109  tickets
0xd820a610ddb18dc4f54aad3c822045fd06cd5d0b  has  104  tickets
0x8f94b4f175298ab637f1b963a65e7fa958d2770d  has  104  tickets
0xf01e34f541caa4a0a1fee65fa55bbf4c19869370  has  91  tickets
0xe038cd04b17130a29fa82fc13d97df2f88b0bf61  has  83  tickets
0xfe2b17345de9fa23a7d64406b9d3146946edb125  has  78  tickets
0x577045c486847fa7bed968d99fa71cf43207a2e9  has  71  tickets
0x873aea03ea1d1db7dba59b74ce7942087ee30e12  has  69  tickets
0x83c42e8cc244c9f9f760b57d3fb7e5f10608119b  has  68  tickets
0x32220e7c4e7448211cd2cd45216bd4cf2e737dea  has  61  tickets
0x6aec90e7a10986c6971439784186521e57f5f4cd  has  54  tickets
0x0cdee0d8d79380e909be5574ba05962df50039da  has  51  tickets
0x8a7ec7b98ec2fbf67c131605868edc5288099005  has  49  tickets
0x92fd8ad0a0567d8b07f9e0e437f5728d3dbd79fd  has  46  tickets
```

getBlockReward()

2.1.3 getBlockReward

def getBlockReward(self, block_identifier='latest'): """Get the block reward for a block

Args:

block_idenfifier: blockNo (int), 'latest', 'earliest', or 'pending'

Returns: block_reward (int) in Wei

“””

```
# Get the block reward

block_reward = web3fsn.getBlockReward()

print('\nThe block reward for the latest block was ',web3fsn.fromWei(block_reward,
↪'ether'),' FSN\n')

input('Hit a key to continue ')
```

```
>>>
The block reward for the latest block was  2.500043904  FSN
```

buyRawTicket()

2.1.4 buyRawTicket

def buyRawTicket(self, transaction): “””Buy a ticket using the raw method (transaction signed and not using IPC)

Args:

transaction (dict): ‘from’: pub_key, ‘nonce’: nonce

Returns: TxHash (hex str)

“””

Here is an example of the function usage

```
nonce = web3fsn.getTransactionCount(pub_key)   # Get the nonce for the wallet

# Construct the transaction
#

transaction = {
    'from':              pub_key,
    'nonce':             nonce,
}

TxHash = web3fsn.buyRawTicket(transaction)

#
print('Transaction hash = ',TxHash)
#
# We can optionally wait for the transaction to occur and block execution until it␣
↪has done so, or times out after timeout seconds
print('Waiting for transaction to go through...')
web3fsn.waitForTransactionReceipt(TxHash, timeout=20)
#
#
res = web3fsn.getTransaction(TxHash)
#
print(res)
#
```

allTickets()

2.1.5 allTickets

def allTickets(self, block_identifier): “”” Return information on all tickets at a certain block height

Args:

block_idenfifier: blockNo (int), ‘latest’, ‘earliest’, or ‘pending’

Returns:

tickets (list of dictionaries): [{‘Owner’: pub_key (address str), ‘Height’: block_height(int), ‘StartTime’: datetime, ‘ExpireTime’: datetime},]

or None (raises BlockNotFound exception)

“””

Here is an example of the function usage

```
# Print out details of all tickets at current block height

allTckts = web3fsn.allTickets('latest')

#print(allTckts)



for a in allTckts:
    tck = allTckts[a]
    st = datetime.fromtimestamp(tck.StartTime).strftime('%c')
    ex = datetime.fromtimestamp(tck.ExpireTime).strftime('%c')
    print('Owner: ',tck.Owner,' Block Height: ',tck.Height,' Start Time: ',st,'␣
↪Expiry Time: ',ex)


print('\n\nTotal number of tickets = ',len(allTckts))

print('\n\nor using totalNumberOfTickets = ',web3fsn.totalNumberOfTickets())
```

Output:-

```
>>>
Owner:  0xced7849e100c92768bebda4575db63301f5515e2  Block Height:  932667  Start␣
↪Time:  Mon Nov 18 12:51:14 2019  Expiry Time:  Wed Dec 18 12:51:14 2019
Owner:  0x8f94b4f175298ab637f1b963a65e7fa958d2770d  Block Height:  937746  Start␣
↪Time:  Tue Nov 19 07:16:32 2019  Expiry Time:  Thu Dec 19 07:16:32 2019
Owner:  0x8f94b4f175298ab637f1b963a65e7fa958d2770d  Block Height:  935119  Start␣
↪Time:  Mon Nov 18 21:43:25 2019  Expiry Time:  Wed Dec 18 21:43:25 2019
Owner:  0xced7849e100c92768bebda4575db63301f5515e2  Block Height:  936422  Start␣
↪Time:  Tue Nov 19 02:28:36 2019  Expiry Time:  Thu Dec 19 02:28:36 2019
Owner:  0x494a792d704e24309fd778641683502fd30f9913  Block Height:  934627  Start␣
↪Time:  Mon Nov 18 19:53:32 2019  Expiry Time:  Wed Dec 18 19:53:32 2019
Owner:  0x6680871ca9c0d0936fe8f875833709381b53e588  Block Height:  938503  Start␣
↪Time:  Tue Nov 19 10:01:13 2019  Expiry Time:  Thu Dec 19 10:01:13 2019
Owner:  0xfe2b17345de9fa23a7d64406b9d3146946edb125  Block Height:  937625  Start␣
↪Time:  Tue Nov 19 06:40:53 2019  Expiry Time:  Thu Dec 19 06:40:53 2019
Owner:  0x3ee9acfaa487a816ed279945166d04c806b82b3e  Block Height:  937642  Start␣
↪Time:  Tue Nov 19 06:53:54 2019  Expiry Time:  Thu Dec 19 06:53:54 2019
Owner:  0x37afe6319dbd980741cd3bfe701f196694d20564  Block Height:  938341  Start␣
↪Time:  Tue Nov 19 09:25:57 2019  Expiry Time:  Thu Dec 19 09:25:57 2019
```

(continues on next page)

(continued from previous page)

```
Owner:  0x76c2ae4281fe1ee1a79ccbdda2516d4d7eb0eb37  Block Height:  937771  Start␣
↪Time:  Tue Nov 19 07:21:58 2019  Expiry Time:  Thu Dec 19 07:21:58 2019
Owner:  0x19f2ca673faaaab7cd28b1c21d466097c3bf8e32  Block Height:  936858  Start␣
↪Time:  Tue Nov 19 04:03:20 2019  Expiry Time:  Thu Dec 19 04:03:20 2019
Owner:  0xfe354642776310a10049b0d90ad2ccad3b12c5ab  Block Height:  937054  Start␣
↪Time:  Tue Nov 19 04:46:00 2019  Expiry Time:  Thu Dec 19 04:46:00 2019
Owner:  0xfe2b17345de9fa23a7d64406b9d3146946edb125  Block Height:  935018  Start␣
↪Time:  Mon Nov 18 21:22:53 2019  Expiry Time:  Wed Dec 18 21:22:53 2019


Total number of tickets =  4588


or using totalNumberOfTickets =  4588
```

totalNumberOfTickets()

2.1.6 totalNumberOfTickets

def totalNumberOfTickets(self, block_identifier=None): “”” Get the total number of tickets at a block height

Args: block_identifier (int), ‘latest’, ‘earliest’, or ‘pending’

Returns: totalNoTickets (int)

ticketsByAddress()

2.1.7 ticketsByAddress

def ticketsByAddress(self, account, block_identifier=None): “”” Retrieve all tickets for a given address

Args: account (public key address str), block_identifier (int), ‘latest’, ‘earliest’, or ‘pending’

Returns:

tickets (list of dictionaries): [{‘Height’: block_height(int), ‘StartTime’: datetime, ‘ExpireTime’: datetime},]

or None (raises BlockNotFound exception)

“””

Here is an example of the function usage

```
Tckts = web3fsn.ticketsByAddress(pub_key)

#print(Tckts)

print('Total number of tickets: ',len(Tckts))
print('\nor using totalNumberOfTicketsByAddress: ',web3fsn.
↪totalNumberOfTicketsByAddress(pub_key),'\n')

for a in Tckts:
    tck = Tckts[a]
    st = datetime.fromtimestamp(tck.StartTime).strftime('%c')
    ex = datetime.fromtimestamp(tck.ExpireTime).strftime('%c')
    print('Block Height: ',tck.Height,' Start Time: ',st,' Expiry Time: ',ex)
```

Output:-

```
>>>
Block Height:  930018  Start Time:  Mon Nov 18 03:14:49 2019  Expiry Time:  Wed Dec␣
↪18 03:14:49 2019
Block Height:  931191  Start Time:  Mon Nov 18 07:30:07 2019  Expiry Time:  Wed Dec␣
↪18 07:30:07 2019
Block Height:  927881  Start Time:  Sun Nov 17 19:29:47 2019  Expiry Time:  Tue Dec␣
↪17 19:29:47 2019
Block Height:  937145  Start Time:  Tue Nov 19 05:05:52 2019  Expiry Time:  Thu Dec␣
↪19 05:05:52 2019
Block Height:  936443  Start Time:  Tue Nov 19 02:33:09 2019  Expiry Time:  Thu Dec␣
↪19 02:33:09 2019
Block Height:  925714  Start Time:  Sun Nov 17 11:38:26 2019  Expiry Time:  Tue Dec␣
↪17 11:38:26 2019
```

totalNumberOfTicketsByAddress()

2.1.8 totalNumberOfTicketsByAddress

def totalNumberOfTicketsByAddress(self, account, block_identifier=None): “””Output the number of tickets for an address

Args: account (hex str) Public key, block_identifier (int), ‘latest’, ‘earliest’, or ‘pending’

Returns: totalTickets (int)

“””

isAutoBuyTicket()

2.1.9 isAutoBuyTicket

def isAutoBuyTicket(self): “””Check to see if tickets are automatically bought for this account (if sufficient balance exists)

Args: None

Returns: isAuto (bool)

“””

startAutoBuyTicket()

2.1.10 startAutoBuyTicket

def startAutoBuyTicket(self): “””Start to auto buy tickets for your account

Args: None

Returns: None

“””

stopAutoBuyTicket()

2.1.11 stopAutoBuyTicket

def stopAutoBuyTicket(self): “””Stop to auto buy tickets for your account

Args: None

Returns: None

“””

2.2 Transactions

For all write transactions, you may optionally specify the ‘gas’ and/or the ‘gasLimit’. You may set ‘gas’: ‘default’ to use the hardcoded value in the class definition.

`getAllBalances``()`

2.2.1 getAllBalances

def getAllBalances(self, account, block_identifier=None): “””Get the balances of all non-timelocked assets for an account

Args: account (hex str) Public key, block_identifier (int), ‘latest’, ‘earliest’, or ‘pending’

Returns: bal_info (dict) key, value pairs for each asset of asset ID and balance

“””

```
>>> bal_info = web3fsn.getAllBalances(pub_key)
>>> for key, val in bal_info.items():
>>>     print(key, val)

0x0e437e96f105776f7f3f96e01ec9def69a6e66ac37d6560b23181350050238f1 95
0x15805e688c7516b8cf005fcb3496cf1e904c4d2579955500f5a18a7957a9d59b 1990
0x34ab2db7e4e5a69e5ec1441d580b9e9599e806cbecf821b87bf4a5952e27ee21 1930
0x3ddec7217915b0c145da683402cfbb94c1b160d23a432f75a39e33e2db091437 1880
0x54cbfda5d4cb46ef1f63d6642f561dcd38dec9fa27a68a0408e9b2b17cc5cfc7 1880
0x5fd3f254ae34bf9bf9dc46f72e4fbbc75844dbe6823f970fa3f7aaedb2925ff6 17
0x6fe2a4955f1424b72627a81a105d483720630e70fc4743182d874c9acc6d5647 99
0xcc966efc1aed2a70d602e9718d528f88cfe304cb91d89338d7f1fe1db3266590 90
0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff␣
↪62866649839999999971
```

`getTransaction``()`

2.2.2 getTransaction

def getTransaction(self, TxHash): “””

Args: TxHash (hex str) Transaction hash

“””

Example of usage :-

```
#
res = web3fsn.getTransaction(TxHash)
#
#print(res)
#
print('\nResults from the transaction :\n')
print('Block number: ',res["blockNumber"])
print('From         : ',res["from"])
print('To           : ',res["to"])
print('Value        : ',web3fsn.fromWei(res["value"],'ether'),' FSN')
print('Gas price    : ',web3fsn.fromWei(res["gasPrice"],'gwei'),' gwei')
```

Output :-

```
>>>
Block number:  891233
From         :  0x7fbFa5679411a97bb2f73Dd5ad01Ca0822FaD9a6
To           :  0xaa8c70e134a5A88aBD0E390F2B479bc31C70Fee1
Value        :  0.02  FSN
Gas price    :  21  gwei
```

getTransactionAndReceipt()

2.2.3 getTransactionAndReceipt

def getTransactionAndReceipt(self, TxHash): “””

> **Args:** TxHash (hex str) Transaction hash
>
> **Returns:** txData (dict) The transaction data and a receipt
>
> “””

Example output:

```
  >>>print(web3fsn.getTransactionAndReceipt(
↪'0x8700056ef2896b47760e661902b21d8f294a80bff87c7e4108d7bbd5bce4ce6d'))

  "txData": {
  "fsnTxInput": {
    "FuncType": "SendAssetFunc",
    "FuncParam": {
      "AssetID": "0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff",
      "To": "0x37a200388caa75edcc53a2bd329f7e9563c6acb6",
      "Value": 1e+18
    }
  },
  "tx": {
    "blockHash": "0xd8d4b5f054cb398b1f0b5bb5d4add5e80d10a432f2c15226f620609577536b6b",
    "blockNumber": "0xad1a5",
    "from": "0x0122bf3930c1201a21133937ad5c83eb4ded1b08",
    "gas": "0x15f90",
    "gasPrice": "0x3b9aca00",
    "hash": "0x8700056ef2896b47760e661902b21d8f294a80bff87c7e4108d7bbd5bce4ce6d",
    "input":
↪"0xf84402b841f83fa0ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff9437a200388caa75
↪",
```

(continues on next page)

(continued from previous page)


```
    "nonce": "0xa7cc",
    "to": "0xffffffffffffffffffffffffffffffffffffffff",
    "transactionIndex": "0x2",
    "value": "0x0",
    "v": "0x16ce3",
    "r": "0x8244e44f720023b240faafab08bb401b1b3167087f2882fa6b8f4fc87b59bdfc",
    "s": "0x277635df431668f4a8b8c8b0702077634dd404db9a1139539d3c276651d3d1ce"
  },
  "receipt": {
    "blockHash": "0xd8d4b5f054cb398b1f0b5bb5d4add5e80d10a432f2c15226f620609577536b6b",
    "blockNumber": "0xad1a5",
    "contractAddress": null,
    "cumulativeGasUsed": "0x10f60",
    "from": "0x0122bf3930c1201a21133937ad5c83eb4ded1b08",
    "fsnLogData": {
      "AssetID": "0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff",
      "To": "0x37a200388caa75edcc53a2bd329f7e9563c6acb6",
      "Value": 1e+18
    },
    "fsnLogTopic": "SendAssetFunc",
    "gasUsed": "0x63e0",
    "logs": [
      {
        "address": "0xffffffffffffffffffffffffffffffffffffffff",
        "topics": [
          "0x0000000000000000000000000000000000000000000000000000000000000002"
        ],
        "data":
↪"0x7b22417373657744944223a223078666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666
↪",
        "blockNumber": "0xad1a5",
        "transactionHash":
↪"0x8700056ef2896b47760e661902b21d8f294a80bff87c7e4108d7bbd5bce4ce6d",
        "transactionIndex": "0x2",
        "blockHash":
↪"0xd8d4b5f054cb398b1f0b5bb5d4add5e80d10a432f2c15226f620609577536b6b",
        "logIndex": "0x2",
        "removed": false
      }
    ],
    "logsBloom":
↪"0x04000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "status": "0x1",
    "to": "0xffffffffffffffffffffffffffffffffffffffff",
    "transactionHash":
↪"0x8700056ef2896b47760e661902b21d8f294a80bff87c7e4108d7bbd5bce4ce6d",
    "transactionIndex": "0x2"
  },
  "receiptFound": true
}
```


getTransactionByBlockNumberAndIndex()

2.2.4 getTransactionByBlockNumberAndIndex

def getTransactionByBlockNumberAndIndex(self, index, block_identifier=None): “””Get transactions from a block with a particular index

Args: index: (int) Index starting at 0, block_identifier (int), ‘latest’, ‘earliest’, or ‘pending’,

Returns: blockinfo (dict) See documentation from web3.py, since this function simply redirects to that

“””

block_identifier (int), ‘latest’, ‘earliest’, or ‘pending’

sendTransaction()

2.2.5 sendTransaction

def sendTransaction(self,transaction): “””Send FSN tokens to another wallet. You can use this method if you have an unlocked wallet (IPC method)

Args: transaction :

‘from’ : pub_key_sender (hex str), ‘to’ : pub_key_receiver (hex str), ‘nonce’ : nonce (int), ‘value’ : value (int) number of FSN * (10**18),

Returns: TxHash transaction hash (hex str)

“””

sendRawTransaction()

2.2.6 sendRawTransaction

def sendRawTransaction(self,transaction, prepareOnly=False): “””Send FSN tokens to another wallet. You can use this method if you have a locked wallet, with a private key, or password

Args: transaction :

‘from’ : pub_key_sender (hex str), ‘to’ : pub_key_receiver (hex str), ‘nonce’ : nonce (int), ‘value’ : value (int) number of FSN * (10**18),

prepareOnly flag (bool) set to True to defer transaction signing to a later point

Returns: TxHash transaction hash (hex str). If prepareOnly=True, then return a Tx_dict (dict)

“””

Here is an example of the function usage

```
value = web3fsn.toWei(0.02,'ether')     # How much FSN are we sending?

nonce = web3fsn.getTransactionCount(pub_key_sender)   # Get the nonce for the sending␣
↪wallet

# Construct the transaction

transaction = {
```

(continues on next page)

(continued from previous page)

```
            "from"  : pub_key_sender,
            "to"    : pub_key_receiver,
            "nonce" : nonce,
            "value" : value,
}

# Send the raw transaction.
TxHash = web3fsn.sendRawTransaction(transaction)
#
#
print('TxHash = ',web3fsn.toHex(TxHash))
#
# We can optionally wait for the transaction to occur and block execution until it␣
↪has done so, or times out after timeout seconds
print('Waiting for transaction to go through...')
web3fsn.waitForTransactionReceipt(TxHash, timeout=20)
#
```

getTransactionCount()

2.2.7 getTransactionCount

def getTransactionCount(self, pub_key): “””Get the next unused transaction ID for this public key

Args: pub_key (hex str)

Returns: nonce (int) The next unused transaction ID for this public key

“””

See *sendRawTransaction* for an example of usage

waitForTransactionReceipt()

2.2.8 waitForTransactionReceipt

def waitForTransactionReceipt(self, TxHash, timeout): “””Check to see that a transaction occured on the blockchain

Args: TxHash (hex str) the transaction hash

timeout (int) Optional timeout in seconds to block program execution and wait for waitForTransactionReceipt

“””

See *sendRawTransaction* for an example of usage

signAndTransmit()

2.2.9 signAndTransmit

def signAndTransmit(self, Tx_dict): “””Sign and transmit a raw transaction on the blockchain

Args: Tx_dict: (dict) transaction

“””

See *Offline Transactions* for an example of usage

2.3 Assets

For all write transactions, you may optionally specify the 'gas' and/or the 'gasLimit'. You may set 'gas': 'default' to use the hardcoded value in the class definition.

getAsset()

2.3.1 getAsset

def getAsset(self, assetId, block_identifier=None): """ Retrieve asset from the blockchain with its asset block_identifier The asset need not be 'enabled' and 'whiteListEnabled' in the fsnapi

Args: assetId (hex str) Hex string asset idenfifier, block_identifier (int), 'latest', 'earliest', or 'pending'

Returns: assetInfo (dict)

"""

Here is an example of the function usage

```
#
#
#asset_name = 'FSN'
asset_name = 'TST5'
blockNo = 'latest'
#
#
asset_Id = web3fsn.getAssetId(asset_name)
print('asset_Id = ',asset_Id)
#
if asset_Id != None:
    asset_dict = web3fsn.getAsset(asset_Id,blockNo)
#
#   print(asset_dict,'\n')
#
```

Outputs :-

```
>>>assetInfo
{ID  :     0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff,
Owner  :     0x0000000000000000000000000000000000000000,
Name  :     Fusion,
Symbol  :     FSN,
Decimals  :     18,
Total  :     81920000000000000000000000,
CanChange  :     False,
Description  :     https://fusion.org
}
```

createAsset()

2.3.2 createAsset

def createAsset(self, transaction):

“”””**Create asset token on the blockchain. You can use this method if you have an unlocked wallet** and are using the IPC mode

Args:

transaction (dict): ‘from’: pub_key (hex str), ‘name’: asset_description (str), ‘nonce’: nonce (int), ‘symbol’: asset_name (str), ‘decimals’: decimal_places (int), ‘total’: total_supply (int), ‘canChange’: change (bool)

Returns: TxHash transaction hash (hex str)

“”””

createRawAsset()

2.3.3 createRawAsset

def createRawAsset(self, transaction, prepareOnly=False):

“”””Create asset token on the blockchain. You can use this method if you have a locked wallet, with a private key, or password

Args:

transaction (dict): ‘from’: pub_key (hex str), ‘name’: asset_description (str), ‘nonce’: nonce (int), ‘symbol’: asset_name (str), ‘decimals’: decimal_places (int), ‘total’: total_supply (int), ‘canChange’: change (bool)

prepareOnly flag (bool) set to True to defer transaction signing to a later point.

Returns: TxHash transaction hash (hex str). If prepareOnly=True, the return a Tx_dict (dict)

“”””

Here is an example of the function usage

```
nonce = web3fsn.getTransactionCount(pub_key)   # Get the nonce for the wallet

# Construct the transaction

transaction = {
  "from":       pub_key,
  "name":       "TestCoin",
  "nonce":       nonce,
  "symbol":     "TST5",
  "decimals":   1,
  "total":      2000,
  "canChange":  True,
}

TxHash = web3fsn.createRawAsset(transaction)

#
print('Transaction hash = ',TxHash)
#
```

incAsset()

2.3.4 incAsset

def incAsset(self, transaction):

“””Increment the supply of an asset. You can use this method if you have an unlocked wallet and are using the IPC mode

Args:

transaction (dict): ‘from’: pub_key_sender (hex str), ‘to’: pub_key_receiver (hex str), ‘nonce’: nonce (int), ‘asset’: asset_Id (hex str), ‘value’: number_to_inc (int), ‘transacData’: message (str)

Returns: TxHash transaction hash (hex str)

“””

incRawAsset()

2.3.5 incRawAsset

def incRawAsset(self, transaction, prepareOnly=False): “””Increment the supply of an asset. You can use this method if you have a locked wallet, with a private key, or password

Args:

transaction (dict): ‘from’: pub_key_sender (hex str), ‘to’: pub_key_receiver (hex str), ‘nonce’: nonce (int), ‘asset’: asset_Id (hex str), ‘value’: number_to_inc (int), ‘transacData’: message (str)

prepareOnly flag (bool) set to True to defer transaction signing to a later point.

Returns: TxHash transaction hash (hex str). If prepareOnly=True, the return a Tx_dict (dict)

“””

Here is an example of the function usage

```
asset_Id = '0x5fd3f254ae34bf9bf9dc46f72e4fbbc75844dbe6823f970fa3f7aaedb2925ff6'
number_to_increase = 5  # The number of tokens you wish to increment by and to send

# Find out some information about this asset
asset_dict = web3fsn.getAsset(asset_Id,'latest')
#print(asset_dict)
print('The asset has the symbol ',asset_dict['Symbol'],' and decimals ',asset_dict[
↪'Decimals'], ' and CanChange is set to ',asset_dict['CanChange'])

nonce = web3fsn.getTransactionCount(pub_key_sender)  # Get the nonce for the wallet

# Construct the transaction

transaction = {
  'from':        pub_key_sender,
  'to':          pub_key_receiver,
  'nonce':       nonce,
  'asset':       asset_Id,
  'value':       int(number_to_increase*10**float(asset_dict['Decimals'])),     # This␣
↪is the integer number of the smallest unit that can be increased, defined by the␣
↪decimals of the asset.
  'transacData': 'Huge airdrop',    # Optional message
}
```

(continues on next page)

(continued from previous page)

```
TxHash = web3fsn.incRawAsset(transaction)

#
print('Transaction hash = ',TxHash)
#
```

decAsset()

2.3.6 decAsset

def decAsset(self, transaction, prepareOnly=False): """Decrement the supply of an asset. You can use this method if you have an unlocked wallet (IPC method)

Args:

transaction (dict): 'from': pub_key_sender (hex str), 'to': pub_key_receiver (hex str), 'nonce': nonce (int), 'asset': asset_Id (hex str), 'value': number_to_dec (int), 'transacData': message (str)

Returns: TxHash transaction hash (hex str)

"""

decRawAsset()

2.3.7 decRawAsset

def decRawAsset(self, transaction, prepareOnly=False): """Decrement the supply of an asset. You can use this method if you have a locked wallet, with a private key, or password

Args:

transaction (dict): 'from': pub_key_sender (hex str), 'to': pub_key_receiver (hex str), 'nonce': nonce (int), 'asset': asset_Id (hex str), 'value': number_to_dec (int), 'transacData': message (str)

prepareOnly flag (bool) set to True to defer transaction signing to a later point.

Returns: TxHash transaction hash (hex str). If prepareOnly=True, the return a Tx_dict (dict)

"""

sendAsset()

2.3.8 sendAsset

def sendAsset(self, transaction): """Send an asset to another wallet. You can use this method if you have an unlocked wallet (IPC method).

Args: transaction :

'from': pub_key_sender (hex str), 'to': pub_key_receiver (hex str) **OR** 'toUSAN': usan (int), 'nonce': nonce (int), 'asset': asset_Id (hex str), 'value': val (int) Number of the asset to send * decimals

Returns: TxHash transaction hash (hex str)

"""

sendRawAsset()

2.3.9 sendRawAsset

def sendRawAsset(self, transaction, prepareOnly=False): “””Send an asset to another wallet. You can use this method if you have a locked wallet, with a private key, or password

Args: transaction :

‘from’: pub_key_sender (hex str), ‘to’: pub_key_receiver (hex str) **OR** ‘toUSAN’: usan (int), ‘nonce’: nonce (int), ‘asset’: asset_Id (hex str), ‘value’: val (int) Number of the asset to send * decimals

prepareOnly flag (bool) set to True to defer transaction signing to a later point.

Returns: TxHash transaction hash (hex str). If prepareOnly=True, the return a Tx_dict (dict)

“””

Here is an example of the function usage

```
asset_Id = '0x5fd3f254ae34bf9bf9dc46f72e4fbbc75844dbe6823f970fa3f7aaedb2925ff6'
number_to_transfer = 5  # The number of tokens you wish to send

# Find out some information about this asset
asset_dict = web3fsn.getAsset(asset_Id,'latest')
#print(asset_dict)
print('The asset has the symbol ',asset_dict['Symbol'],' and decimals ',asset_dict[
↪'Decimals'])

nonce = web3fsn.getTransactionCount(pub_key_sender)  # Get the nonce for the wallet

# Construct the transaction

transaction = {
  'from':       pub_key_sender,
  'to':         pub_key_receiver,
  'nonce':      nonce,
  'asset':      asset_Id,
  'value':      int(number_to_transfer*10**float(asset_dict['Decimals'])),     # This␣
↪is the integer number of the smallest unit that can be sent, defined by the␣
↪decimals of the asset.
}

TxHash = web3fsn.sendRawAsset(transaction)

#
print('Transaction hash = ',TxHash)
#
```

2.4 Timelocks

For all write transactions, you may optionally specify the ‘gas’ and/or the ‘gasLimit’. You may set ‘gas’: ‘default’ to use the hardcoded value in the class definition.

getAllTimeLockBalances()

2.4.1 getAllTimeLockBalances

def getAllTimeLockBalances(self, account, block_identifier=None): “””Demonstrate getting timelock information for all assets for a public key

Args: account (hex str) Public key, block_identifier (int), ‘latest’, ‘earliest’, or ‘pending’

Returns: asset_timelocks (list of dicts)

“””

Here is an example of the function usage

```
#
pub_key = '0x3333333333333333333333333333333333333333'
blockNo = 'latest'
#
#
asset_timelocks = web3fsn.getAllTimeLockBalances(pub_key, block_identifier=None)
#
```

```
>>>print(asset_timelocks)
AttributeDict({'0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff':
AttributeDict({'Items':
[AttributeDict({'StartTime': 1576582707, 'EndTime': 1583280000, 'Value':
↪'500000000000000000000'}),
AttributeDict({'StartTime': 1576610988, 'EndTime': 1583280000, 'Value':
↪'500000000000000000000'}),
AttributeDict({'StartTime': 1576638890, 'EndTime': 1581465600, 'Value':
↪'500000000000000000000'}),
AttributeDict({'StartTime': 1576654208, 'EndTime': 1581465600, 'Value':
↪'500000000000000000000'}),
AttributeDict({'StartTime': 1576658926, 'EndTime': 1581465600, 'Value':
↪'500000000000000000000'}),
AttributeDict({'StartTime': 1576676450, 'EndTime': 1581465600, 'Value':
↪'500000000000000000000'}),
AttributeDict({'StartTime': 1576679139, 'EndTime': 1581120000, 'Value':
↪'500000000000000000000'}),
AttributeDict({'StartTime': 1576689626, 'EndTime': 1580083200, 'Value':
↪'500000000000000000000'}),
AttributeDict({'StartTime': 1576694666, 'EndTime': 1578787200, 'Value':
↪'500000000000000000000'}),
AttributeDict({'StartTime': 1576698618, 'EndTime': 1578787200, 'Value':
↪'500000000000000000000'}),
AttributeDict({'StartTime': 1576680247, 'EndTime': 1580083200, 'Value':
↪'260000000000000000000'}),
AttributeDict({'StartTime': 1576691387, 'EndTime': 1578787200, 'Value':
↪'260000000000000000000'}),
AttributeDict({'StartTime': 1576680247, 'EndTime': 1580774400, 'Value':
↪'240000000000000000000'}),
AttributeDict({'StartTime': 1576691387, 'EndTime': 1580083200, 'Value':
↪'240000000000000000000'}),
AttributeDict({'StartTime': 1590421416, 'EndTime': 18446744073709551615, 'Value':
↪'20000'}),
AttributeDict({'StartTime': 1593031827, 'EndTime': 18446744073709551615, 'Value':
↪'10000'})]})})
```

getTimeLockBalance()

2.4.2 getTimeLockBalance

def getTimeLockBalance(self, assetId, account, block_identifier=None): “””Demonstrate getting timelock information about an asset for a public key

Args: assetId (hex str) Hex string asset idenfifier, account (hex str) Public key, block_identifier (int), ‘latest’, ‘earliest’, or ‘pending’

Returns: asset_timelocks (list of dicts)

“””

Here is an example of the function usage

```
#
pub_key = '0x3333333333333333333333333333333333333333'
asset_name = 'FSN'
blockNo = 'latest'
#
#
asset_Id = web3fsn.getAssetId(asset_name)
#
asset_timelocks = web3fsn.getTimeLockBalance(asset_Id, pub_key, blockNo)
```

```
>>>n_items = len(asset_timelocks.Items)
>>>print('\nNumber of timelocked ', asset_name, ' items = ',n_items,'\n')
Number of timelocked  FSN  items =  3
>>>for i in range(n_items):
print('Asset ',i,'\n')
tm = asset_timelocks.Items[i].StartTime
print('Start Time :   ',datetime.fromtimestamp(tm).strftime('%c'))
tm = asset_timelocks.Items[i].EndTime
if tm >= web3fsn.BN():
    endtime = 'Infinity'
else:
    endtime = datetime.fromtimestamp(tm).strftime('%c')
print('End Time   :   ',endtime)
val = int(asset_timelocks.Items[i].Value)
print(web3fsn.fromWei(val,'ether'),' ',asset_name,'\n')

Asset  0

Start Time :    Sun Dec 15 13:30:03 2019
End Time   :    Wed Mar  4 00:00:00 2020
5000   FSN

Asset  1

Start Time :    Tue Dec 17 11:38:27 2019
End Time   :    Wed Mar  4 00:00:00 2020
5000   FSN

Asset  2

Start Time :    Tue Dec 17 19:29:48 2019
End Time   :    Wed Feb 12 00:00:00 2020
5000   FSN
```

sendToTimeLock()

2.4.3 sendToTimeLock

def sendToTimeLock(self, transaction): “””To send asset tokens on the Fusion blockchain to timelock with an unlocked wallet (IPC method) without changing the time lock.

Args:

transaction (dict): ‘from’: pub_key_sender (hex str), ‘to’: pub_key_receiver (hex str) **OR** ‘toUSAN’: usan (int), ‘nonce’: nonce (int), ‘asset’: asset_Id (hex str), ‘value’: nToSend (int), ‘start’: startdate (date str - Optional), ‘end’: enddate (date str - Optional)

Returns: TxHash transaction hash (hex str)

“””

sendToRawTimeLock()

2.4.4 sendToRawTimeLock

def sendToRawTimeLock(self, transaction, prepareOnly=False): “””To send asset tokens on the Fusion blockchain to timelock using the raw transaction method, without changing the time lock.

Args:

transaction (dict): ‘from’: pub_key_sender (hex str), ‘to’: pub_key_receiver (hex str) **OR** ‘toUSAN’: usan (int), ‘nonce’: nonce (int), ‘asset’: asset_Id (hex str), ‘value’: nToSend (int), ‘start’: startdate (date str - Optional), ‘end’: enddate (date str - Optional)

prepareOnly flag (bool) set to True to defer transaction signing to a later point.

Returns: TxHash transaction hash (hex str). If prepareOnly=True, the return a Tx_dict (dict)

“””

Here is an example of the function usage without start and end dates (see *timeLockToRawTimeLock* function below for an example using ‘start’ and ‘end’

```
asset_Id = web3fsn.getAssetId('FSN')
#asset_Id = "0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff"
number_to_transfer = 2  # The number of tokens you wish to send

# Find out some information about this asset
asset_dict = web3fsn.getAsset(asset_Id,'latest')
#print(asset_dict)
asset_name = asset_dict['Symbol']
print('The asset has the symbol ',asset_name,' and decimals ',asset_dict['Decimals'])


nToSend = int(number_to_transfer*10**float(asset_dict['Decimals']))

nonce = web3fsn.getTransactionCount(pub_key_sender)  # Get the nonce for the wallet

# Construct the transaction
#
# Example of valid dates for 'start' and 'end'. Can also use 'now' and 'infinity'.
#"2007-03-01T13:00:00+0100"  or  UTC = "2007-03-01T12:00:00"

transaction = {
  'from':       pub_key_sender,
  'to':         pub_key_receiver,
```

(continues on next page)

(continued from previous page)

```
  'nonce':      nonce,
  'asset':      asset_Id,
  'value':      nToSend,
  #'start':      'now',
  #'end':        '2020-06-01T06:00:59',
  'start':      '2020-12-20T06:01:01',
  'end':        'infinity',
}

TxHash = web3fsn.sendRawTimeLock(transaction)

#
print('Transaction hash = ',TxHash)
#
# We can optionally wait for the transaction to occur and block execution until it␣
↪has done so, or times out after timeout seconds
print('Waiting for transaction to go through...')
web3fsn.waitForTransactionReceipt(TxHash, timeout=20)
#
#
res = web3fsn.getTransaction(TxHash)
#
#print(res)
#
# Show the timelocks for the pub_key_receiver
#
asset_timelocks = web3fsn.getTimeLockBalance(asset_Id, pub_key_receiver, 'latest')
#
n_items = len(asset_timelocks.Items)
print('\nNumber of timelocked ', asset_name, ' = ',n_items,'\n')
#
#
for i in range(n_items):
    print('Asset ',i,'\n')
    tm = asset_timelocks.Items[i].StartTime
    print('Start Time :   ',datetime.fromtimestamp(tm).strftime('%c'))
    tm = asset_timelocks.Items[i].EndTime
    if tm >= web3fsn.BN():
        endtime = 'Infinity'
    else:
        endtime = datetime.fromtimestamp(tm).strftime('%c')
    print('End Time   :   ',endtime)
    val = int(asset_timelocks.Items[i].Value)
    print(val/(10**asset_dict['Decimals']),' ',asset_name,'\n')
#
```

assetToTimeLock()

2.4.5 assetToTimeLock

def assetToTimeLock(self, transaction): “””To send asset tokens on the Fusion blockchain to timelock with an unlocked wallet (IPC method) without changing the time lock.

Args:

transaction (dict): ‘from’: pub_key_sender (hex str), ‘to’: pub_key_receiver (hex str) **OR** ‘toUSAN’: usan (int), ‘nonce’: nonce (int), ‘asset’: asset_Id (hex str), ‘value’: nToSend (int), ‘start’: startdate

(date str - Optional), 'end': enddate (date str - Optional)

Returns: TxHash transaction hash (hex str)

""""

assetToRawTimeLock()

2.4.6 assetToRawTimeLock

def assetToRawTimeLock(self, transaction, prepareOnly=False): """To send asset tokens on the Fusion blockchain to timelock using the raw transaction method, without changing the time lock.

Args:

transaction (dict): 'from': pub_key_sender (hex str), 'to': pub_key_receiver (hex str) **OR** 'toUSAN': usan (int), 'nonce': nonce (int), 'asset': asset_Id (hex str), 'value': nToSend (int), 'start': startdate (date str - Optional), 'end': enddate (date str - Optional)

prepareOnly flag (bool) set to True to defer transaction signing to a later point.

Returns: TxHash transaction hash (hex str). If prepareOnly=True, the return a Tx_dict (dict)

""""

Here is an example of the function usage without start and end dates (see *timeLockToRawTimeLock* function below for an example using 'start' and 'end'

```
asset_Id = '0x3ddec7217915b0c145da683402cfbb94c1b160d23a432f75a39e33e2db091437'
number_to_transfer = 5  # The number of tokens you wish to send

# Find out some information about this asset
asset_dict = web3fsn.getAsset(asset_Id,'latest')
#print(asset_dict)
asset_name = asset_dict['Symbol']
print('The asset has the symbol ',asset_name,' and decimals ',asset_dict['Decimals'])


nToSend = int(number_to_transfer*10**float(asset_dict['Decimals']))

nonce = web3fsn.getTransactionCount(pub_key_sender)  # Get the nonce for the wallet

# Construct the transaction
#
# We do not include start or end times here :-
#
transaction = {
  'from':       pub_key_sender,
  'to':         pub_key_sender,    # send the assets to the same wallet, but as a␣
↪timelock
  'nonce':      nonce,
  'asset':      asset_Id,
  'value':      nToSend,
}

TxHash = web3fsn.assetToRawTimeLock(transaction)

#
print('Transaction hash = ',TxHash)
#
```

(continues on next page)

(continued from previous page)

```
# We can optionally wait for the transaction to occur and block execution until it␣
↪has done so, or times out after timeout seconds
print('Waiting for transaction to go through...')
web3fsn.waitForTransactionReceipt(TxHash, timeout=20)
#
#
res = web3fsn.getTransaction(TxHash)
#
#print(res)
#
#
# Now send them back to assets again
#
    print('\nNow send the time lock tokens back to assets...')
    reply = input('Check your wallet and hit enter to continue > ')
#
    transaction['nonce'] = nonce + 1

# Use the same transaction data
#
TxHash = web3fsn.timeLockToRawAsset(transaction)

#
print('Transaction hash = ',TxHash)
#
# We can optionally wait for the transaction to occur and block execution until it␣
↪has done so, or times out after timeout seconds
print('Waiting for transaction to go through...')
web3fsn.waitForTransactionReceipt(TxHash, timeout=20)
#
#
res = web3fsn.getTransaction(TxHash)
#
```

timeLockToAsset()

2.4.7 timeLockToAsset

def timeLockToAsset(self, transaction): “””To send timelocked asset tokens on the Fusion blockchain to assets with an unlocked wallet (IPC method)

Args:

transaction (dict): ‘from’: pub_key_sender (hex str), ‘to’: pub_key_receiver (hex str) **OR** ‘toUSAN’: usan (int), ‘nonce’: nonce (int), ‘asset’: asset_Id (hex str), ‘value’: nToSend (int)

Returns: TxHash transaction hash (hex str)

“””

timeLockToRawAsset()

2.4.8 timeLockToRawAsset

def timeLockToRawAsset(self, transaction, prepareOnly=False): “””To send timelocked asset tokens on the Fusion blockchain to assets using the raw transaction method, without changing the time lock.

Args:

transaction (dict): 'from': pub_key_sender (hex str), 'to': pub_key_receiver (hex str) **OR** 'toUSAN': usan (int), 'nonce': nonce (int), 'asset': asset_Id (hex str), 'value': nToSend (int)

prepareOnly flag (bool) set to True to defer transaction signing to a later point. If prepareOnly=True, the return a Tx_dict (dict)

Returns: TxHash transaction hash (hex str)

"""

See the example code for *assetToRawTimeLock* for usage

timeLockToTimeLock()

2.4.9 timeLockToTimeLock

def timeLockToTimeLock(self, transaction): """To create a new timelock for an existing timelocked asset with an unlocked wallet (IPC method)

Args:

transaction (dict): 'from': pub_key_sender (hex str), 'to': pub_key_receiver (hex str) **OR** 'toUSAN': usan (int), 'nonce': nonce (int), 'asset': asset_Id (hex str), 'value': nToSend (int), 'start': startdate (date str - Optional), 'end': enddate (date str - Optional)

Returns: TxHash transaction hash (hex str)

"""

timeLockToRawTimeLock()

2.4.10 timeLockToRawTimeLock

def timeLockToRawTimeLock(self, transaction, prepareOnly=False): """To create a new timelock for an existing timelocked asset using the raw transaction method

Args:

transaction (dict): 'from': pub_key_sender (hex str), 'to': pub_key_receiver (hex str) **OR** 'toUSAN': usan (int), 'nonce': nonce (int), 'asset': asset_Id (hex str), 'value': nToSend (int), 'start': startdate (date str - Optional), 'end': enddate (date str - Optional)

prepareOnly flag (bool) set to True to defer transaction signing to a later point.

Returns: TxHash transaction hash (hex str). If prepareOnly=True, the return a Tx_dict (dict)

"""

Here is an example of the function usage

```
asset_Id = '0x3ddec7217915b0c145da683402cfbb94c1b160d23a432f75a39e33e2db091437'
number_to_transfer = 0.4  # The number of tokens you wish to send

# Find out some information about this asset
asset_dict = web3fsn.getAsset(asset_Id,'latest')
#print(asset_dict)
asset_name = asset_dict['Symbol']
print('The asset has the symbol ',asset_name,' and decimals ',asset_dict['Decimals'])
#
```

(continues on next page)

(continued from previous page)

```
#
#  First of all send some token assets in your wallet to timelock

nToSend = int(number_to_transfer*10**float(asset_dict['Decimals']))

nonce = web3fsn.getTransactionCount(pub_key_sender)  # Get the nonce for the wallet

# Construct the transaction
#
# Example of valid dates for 'start' and 'end'. Can also use 'now' and 'infinity'.
#"2007-03-01T13:00:00+0100"  or  UTC = "2007-03-01T12:00:00"

transaction = {
  'from':       pub_key_sender,
  'to':         pub_key_sender,  # same wallet
  'nonce':      nonce,
  'asset':      asset_Id,
  'value':      nToSend,
  'start':      '2020-06-01T06:00:00+0400',
  'end':        'Infinity',
}

TxHash = web3fsn.assetToRawTimeLock(transaction)

#
print('Transaction hash = ',TxHash)
#
# We can optionally wait for the transaction to occur and block execution until it␣
↪has done so, or times out after timeout seconds
print('Waiting for transaction to go through...')
web3fsn.waitForTransactionReceipt(TxHash, timeout=120)
#
#
res = web3fsn.getTransaction(TxHash)
#
#print(res)
#
# Show the timelocks for the pub_key_sender
#
asset_timelocks = web3fsn.getTimeLockBalance(asset_Id, pub_key_sender, 'latest')
#
n_items = len(asset_timelocks.Items)
print('\nNumber of timelocked ', asset_name, ' items = ',n_items,'\n')
#
#
for i in range(n_items):
    print('Asset ',i,'\n')
    tm = asset_timelocks.Items[i].StartTime
    print('Start Time :   ',datetime.fromtimestamp(tm).strftime('%c'))
    tm = asset_timelocks.Items[i].EndTime
    if tm >= web3fsn.BN():
        endtime = 'Infinity'
    else:
        endtime = datetime.fromtimestamp(tm).strftime('%c')
    print('End Time   :   ',endtime)
    val = int(asset_timelocks.Items[i].Value)
    print(val/(10**asset_dict['Decimals']),' ',asset_name,'\n')
```

(continues on next page)

(continued from previous page)

```
#
#
# Now we will send some timelocked tokens to someone else's wallet.
#
#
print('\nNow send the time lock tokens someone else\'s wallet...')
reply = input('Check your wallet and hit enter to continue > ')
#
transaction = {
    'from':       pub_key_sender,
    'to':         pub_key_receiver,
    'nonce':      nonce + 1,
    'asset':      asset_Id,
    'value':      int(nToSend/2),       # Send only half
    'start':      '2020-12-01T06:00:00', # Send a later time portion
    'end':        'Infinity',
}


TxHash = web3fsn.timeLockToRawTimeLock(transaction)

#
print('Transaction hash = ',TxHash)
#
# We can optionally wait for the transaction to occur and block execution until it␣
↪has done so, or times out after timeout seconds
print('Waiting for transaction to go through...')
web3fsn.waitForTransactionReceipt(TxHash, timeout=120)
#
#
res = web3fsn.getTransaction(TxHash)
#
#print(res)
#
# Show the timelocks for the pub_key_sender
#
asset_timelocks = web3fsn.getTimeLockBalance(asset_Id, pub_key_sender, 'latest')
#
n_items = len(asset_timelocks.Items)
print('\nNumber of timelocked ', asset_name, ' items = ',n_items,'\n')
#
#
for i in range(n_items):
    print('Asset ',i,'\n')
    tm = asset_timelocks.Items[i].StartTime
    print('Start Time :   ',datetime.fromtimestamp(tm).strftime('%c'))
    tm = asset_timelocks.Items[i].EndTime
    if tm >= web3fsn.BN():
        endtime = 'Infinity'
    else:
        endtime = datetime.fromtimestamp(tm).strftime('%c')
    print('End Time   :   ',endtime)
    val = int(asset_timelocks.Items[i].Value)
    print(val/(10**asset_dict['Decimals']),' ',asset_name,'\n')
```

Output from this code :-

```
>>>
Number of timelocked  TST1  items =  2

Asset  0

Start Time :    Wed Nov 20 15:15:26 2019
End Time   :    Infinity
3.5   TST1

Asset  1

Start Time :    Wed Nov 20 15:15:26 2019
End Time   :    Tue Dec  1 05:59:59 2020
2.7   TST1


Now send the time lock tokens someone else's wallet...
Check your wallet and hit enter to continue >
Transaction hash =  0x6e13854a339778494580d61c7094f4eec579744f9a86042ffcca7cad3c60b53d
Waiting for transaction to go through...

Number of timelocked  TST1  items =  2

Asset  0

Start Time :    Wed Nov 20 15:15:26 2019
End Time   :    Infinity
3.3   TST1

Asset  1

Start Time :    Wed Nov 20 15:15:26 2019
End Time   :    Tue Dec  1 05:59:59 2020
2.9   TST1
```

2.5 Swaps

For all write transactions, you may optionally specify the 'gas' and/or the 'gasLimit'. You may set 'gas': 'default' to use the hardcoded value in the class definition.

getSwap()

2.5.1 getSwap

def getSwap(self, swapId, block_identifier='latest'): """Get the information about one swap

Args: swapId (hex str), block_identifier (int), 'latest', 'earliest', or 'pending'

Returns: swap_dict (dict) information about the swap.

"ID": (hex str), "Owner": (hex str) public_key, "FromAssetID": (hex str) asset ID, "FromStartTime": (int) seconds since epoch, "FromEndTime": (int) seconds since epoch, "MinFromAmount": (float) tokens*decimals, "ToAssetID": (hex str) asset ID, "ToStartTime": (int) seconds since epoch, "ToEndTime": (int) seconds since epoch, "MinToAmount": (float) tokens*decimals, "SwapSize": (int), "Targes": (list of

hex str) list of private addresses, or [] for none, “Time”: (int) seconds since epoch. Time swap initiated, “Description”: (str), “Notation”: (int) USAN

“””

```
res = web3fsn.getTransaction(TxHash)
#
print(res)
#
# Get some information about the swap
#
swap_dict = web3fsn.getSwap(TxHash)
#
print(swap_dict)
```

makeSwap()

2.5.2 makeSwap

For all write transactions, you may optionally specify the ‘gas’ and/or the ‘gasLimit’. You may set ‘gas’: ‘default’ to use the hardcoded value in the class definition.

def makeSwap(self, transaction): “””Create a swap on the Quantum Swap Market. You can use this method if you have an unlocked wallet (IPC method)

Args: transaction (dict)

‘from’: pub_key_sender (hex str), ‘nonce’: nonce (int), ‘ToAssetID’: assetId (hex str), ‘ToStartTime’: to_st_time (str) can be ‘now’, ‘ToEndTime’: to_en_time (str) can be ‘infinity’, or e.g. ‘2020-06-01T06:00:00+0400’, ‘MinToAmount’: nToSend (str) Minimum No. of tokens*decimals to swap, ‘FromAssetID’: assetId (hex str), ‘FromStartTime’: from_st_time (str) Defaults to ‘now’ ‘FromEndTime’: from_en_time (str) Defaults to ‘infinity’ ‘MinFromAmount’: nToReceive (int), Minimum No. tokens to receive * decimals|br| ‘SwapSize’: swap_size (int) swap size, ‘Targes’: target wallets (list), # Leave as an empty list [] for a public swap.

Returns: TxHash transaction hash (hex str)

“””

makeRawSwap()

2.5.3 makeRawSwap

def makeRawSwap(self, transaction, prepareOnly=False): “””Create a swap on the Quantum Swap Market. You can use this method if you have a locked wallet, with a private key, or password

Args: transaction (dict)

‘from’: pub_key_sender (hex str), ‘nonce’: nonce (int), ‘ToAssetID’: assetId (hex str), ‘ToStartTime’: to_st_time (datetime) can be ‘now’, ‘ToEndTime’: to_en_time (datetime) can be ‘infinity’, or e.g. ‘2020-06-01T06:00:00+0400’, ‘MinToAmount’: nToSend (int) Minimum No. of tokens*decimals to swap, ‘FromAssetID’: assetId (hex str), ‘FromStartTime’: from_st_time (datetime) Defaults to ‘now’ ‘FromEndTime’: from_en_time (datetime) Defaults to ‘infinity’ ‘MinFromAmount’: nToReceive (int), Minimum No. tokens to receive * decimals|br| ‘SwapSize’: swap_size (int) swap size, ‘Targes’: target wallets (list), # Leave as an empty list [] for a public swap.

prepareOnly flag (bool) set to True to defer transaction signing to a later point.

Returns: TxHash transaction hash (hex str). If prepareOnly=True, the return a Tx_dict (dict)

“””

```
assetId_TST1 = '0x3ddec7217915b0c145da683402cfbb94c1b160d23a432f75a39e33e2db091437'
assetId_TST2 = '0x34ab2db7e4e5a69e5ec1441d580b9e9599e806cbecf821b87bf4a5952e27ee21'

number_to_swap = 5      # The minimum number of tokens you wish to swap
number_to_receive = 1   # The minimum number of tokens you wish to receive

# Find out some information about these assets
asset_dict = web3fsn.getAsset(assetId_TST1,'latest')
asset_TST1_name = asset_dict['Symbol']
asset_TST1_decimals = asset_dict['Decimals']

asset_dict = web3fsn.getAsset(assetId_TST2,'latest')
asset_TST2_name = asset_dict['Symbol']
asset_TST2_decimals = asset_dict['Decimals']
#
#

nToSend = int(number_to_swap*10**float(asset_TST1_decimals))
nToReceive = int(number_to_receive*10**float(asset_TST2_decimals))

nonce = web3fsn.getTransactionCount(pub_key_sender)  # Get the nonce for the wallet

# Construct the transaction
#
# Example of valid dates for 'start' and 'end'. Can also use 'now' and 'infinity',
# or a hex string with the number of seconds since "1970-01-01T00:00:00+0000".
# "2007-03-01T13:00:00+0100"  or  UTC = "2007-03-01T12:00:00"

transaction = {
    'from':                 pub_key_sender,
    'nonce':                nonce,
    'ToAssetID':            assetId_TST1,
    'ToStartTime':          'now',
    'ToEndTime':            '2020-06-01T06:00:00+0400',
    'MinToAmount':          nToSend,
    'FromAssetID':          assetId_TST2,
    #'FromStartTime':       Defaults to 'now'
    #'FromEndTime':         Defaults to 'infinity'
    'MinFromAmount':        nToReceive,
    'SwapSize':             3,
    'Targes':               [],  # Leave as an empty list [] for a public swap.
}

TxHash = web3fsn.makeRawSwap(transaction)

#
print('Transaction hash = ',TxHash)
#
# We can optionally wait for the transaction to occur and block execution until it␣
↪has done so, or times out after timeout seconds
print('Waiting for transaction to go through...')
web3fsn.waitForTransactionReceipt(TxHash, timeout=20)
#
#
res = web3fsn.getTransaction(TxHash)
```

(continues on next page)

(continued from previous page)

```
#
```

makeRawMultiSwap()

2.5.4 makeRawMultiSwap

def makeRawMultiSwap(self, transaction, prepareOnly=False): “””Create a multi swap on the Quantum Swap Market. You can use this method if you have a locked wallet, with a private key, or password

Args: transaction (dict) :

‘from’: pub_key_sender (hex str), ‘nonce’: nonce (int), ‘ToAssetID’: assetId (list of hex str), ‘ToStartTime’: to_st_time (datetime), optional, can be ‘now’, ‘ToEndTime’: to_en_time (datetime) can be ‘infinity’, or e.g. ‘2020-06-01T06:00:00+0400’, ‘MinToAmount’: nToSend (list of int) Minimum No. of tokens*decimals to swap, ‘FromAssetID’: assetId (list of hex str), ‘FromStartTime’: from_st_time (list of datetime) Defaults to ‘now’ ‘FromEndTime’: from_en_time (list of datetime) Defaults to ‘infinity’ ‘MinFromAmount’: nToReceive (list of int), Minimum No. tokens to receive * decimals|br| ‘SwapSize’: swap_size (int) swap size, ‘Targes’: target wallets (list), # Leave as an empty list [] for a public swap.

prepareOnly flag (bool) set to True to defer transaction signing to a later point.

Returns: TxHash transaction hash (hex str). If prepareOnly=True, the return a Tx_dict (dict)

“””

recallSwap()

2.5.5 recallSwap

def recallSwap(self, transaction): “”” Recall a swap from the Quantum Swap Market. You can use this method if you have an unlocked wallet (IPC method)

Args: transaction (dict)

‘from’: pub_key_sender, ‘nonce’: nonce (int), ‘SwapID’: swapId (hex str)

Returns: TxHash transaction hash (hex str)

“””

recallRawSwap()

2.5.6 recallRawSwap

def recallRawSwap(self, transaction, prepareOnly=False): “”” Recall a swap from the Quantum Swap Market. You can use this method if you have a locked wallet, with a private key, or password

Args: transaction (dict)

‘from’: pub_key_sender, ‘nonce’: nonce (int), ‘SwapID’: swapId (hex str)

prepareOnly flag (bool) set to True to defer transaction signing to a later point

Returns: TxHash transaction hash (hex str). If prepareOnly=True, the return a Tx_dict (dict)

“””

Example code :-

```
#
transaction = {
    'from':                 pub_key_sender,
    'nonce':                nonce + 1,
    'SwapID':               swap,
}

TxHash = web3fsn.recallRawSwap(transaction)
```

takeSwap()

2.5.7 takeSwap

def takeSwap(self, transaction): """Take a swap on the Quantum Swap Market. You can use this method if you have an unlocked wallet (IPC method)

Args: transaction (dict)

'from': pub_key (hex str), 'nonce': nonce (int), 'SwapID': swapHash (hex str), 'Size': number_to_receive (int)

Returns: TxHash transaction hash (hex str)

"""

takeRawSwap()

2.5.8 takeRawSwap

def takeRawSwap(self, transaction, prepareOnly=False): """Take a swap on the Quantum Swap Market. You can use this method if you have a locked wallet, with a private key, or password

Args: transaction (dict)

'from': pub_key (hex str), 'nonce': nonce (int), 'SwapID': swapHash (hex str), 'Size': number_to_receive (int)

prepareOnly flag (bool) set to True to defer transaction signing to a later point.

Returns: TxHash transaction hash (hex str). If prepareOnly=True, then return a Tx_dict (dict)

"""

Example code :-

```
swapInfo = web3fsn.getAllSwaps()

#print(swapInfo)

for swp in swapInfo:
    print('SwapID : {} From asset : {} To asset : {} Size : {} MinFromAmount : {}␣
↪MinToAmount : {} SwapSize : {}\n'
          .format(swp['swapID'],swp['fromAsset'],swp['toAsset'],swp['size'],
          swp['MinFromAmount'],swp['MinToAmount'],swp['SwapSize']))

swapID = input('Enter a swapID from the above list  ')
```

(continues on next page)

(continued from previous page)

```
number_to_receive = input('Enter the number you wish to receive ')    # The number of␣
↪tokens you wish to receive


nonce = web3fsn.getTransactionCount(pub_key)  # Get the nonce for the wallet

# Construct the transaction
#

transaction = {
    'from':             pub_key,
    'nonce':            nonce,
    'SwapID':           swapID,
    'Size':             number_to_receive,
}

TxHash = web3fsn.takeRawSwap(transaction)

#
print('Transaction hash = ',TxHash)
```

2.6 Notation (USAN)

For all write transactions, you may optionally specify the 'gas' and/or the 'gasLimit'. You may set 'gas': 'default' to use the hardcoded value in the class definition.

getNotation()

2.6.1 getNotation

def getNotation(self, account, block_identifier=None): """Get the USAN for a public key address

Args: account (hex str) Public key, block_identifier (int), 'latest', 'earliest', or 'pending'

Returns: usan (int)

"""

See *genRawNotation* for an example of usage

getLatestNotation()

2.6.2 getLatestNotation

def getLatestNotation(self, account, block_identifier=None): """Get the last notation on the blockchain

Args: account (hex str) Public key, block_identifier (int), 'latest', 'earliest', or 'pending'

Returns: usan (int)

"""

getAddressByNotation()

2.6.3 getAddressByNotation

def getAddressByNotation(self, notation, block_identifier=None): """Get the public key corresponding to a USAN

Args: notation (int) USAN block_identifier (int), 'latest', 'earliest', or 'pending'

Returns: pub_key (hex str)

"""

See *genRawNotation* for an example of usage

genRawNotation()

2.6.4 genRawNotation

def genRawNotation(self,transaction, prepareOnly=False): """Generate a new USAN for an account

Args: transaction (dict) :

'from': pub_key_sender (hex str), 'nonce': nonce (int)

prepareOnly flag (bool) set to True to defer transaction signing to a later point.

Returns: TxHash transaction hash (hex str). If prepareOnly=True, then return a Tx_dict (dict)

"""

Example code :-

```
nonce = web3fsn.getTransactionCount(pub_key_sender)  # Get the nonce for the wallet

# Construct the transaction

transaction = {
  'from':       pub_key_sender,
  'nonce':      nonce,
}

TxHash = web3fsn.genRawNotation(transaction)


#
print('Transaction hash = ',TxHash)
#
# We can optionally wait for the transaction to occur and block execution until it␣
↪has done so, or times out after timeout seconds
print('Waiting for transaction to go through...')
web3fsn.waitForTransactionReceipt(TxHash, timeout=20)
#
#
res = web3fsn.getTransaction(TxHash)
#
#print(res)
#
#
# Request the value back
#
```

(continues on next page)

(continued from previous page)

```
notation = web3fsn.getNotation(pub_key_sender)
#
print('The generated notation is ',notation)
#
# Check that this notation refers to our public key
#
pubk = web3fsn.getAddressByNotation(notation)
#
print('The public address is ',pubk)
#
```

2.7 Fusion API

There is a centralized service generating data about Fusion's blockchain. The output is served via an express server in JSON format. This can provide quick access to important data without having to scan the whole blockchain youself to compile it. It is possible that the format of the output will change with time (or may even cease), but below you can find some functions currently to access various parts of the data in a format useful for application development.

fsnprice()

2.7.1 fsnprice

def fsnprice(self): """Information about the current price, market capitation and circulating supply of the Fusion token

Returns: fsnInfo (dict)

"""

getAllSwaps()

2.7.2 getAllSwaps

def getAllSwaps(self, pageNo): """Get information on all current swaps from fsnapi

Args: PageNo (int) The data is served with 100 records per page, starting at page 0. Simply increment until the list is exhausted and the length of the output is less than 100.

Returns: swap_dict (dict) with fields :-

'swapID' (hex str) Can be used for taking swaps etc. 'timeStamp' (str) date and time swap appears on chain 'fromAddress' (hex str) public pub_key 'fromAsset' (hex str) assetId 'toAsset' (hex str) assetId 'recCreated' (str) date and time 'height' (int) block height swap recCreated 'hash' (hex str) transaction hash for swap creation 'size' (int) Total swap size 'Description' (str) 'FromStartTime' (str) start of swap timelock 'ToEndTime' (str) end of swap timelock 'MinFromAmount'(int) minimum amount for the swap - from 'MinToAmount' (int) minumum amount for the swap - to 'SwapSize' (int) 'Targes' (list) Target wallets for private swaps 'Time' (str) date and time swap created 'ToAssetID' (hex str) to assetId

"""

Example code

```
pageNo = 0   # Only get the most recent 100 records on the first page

swap_dict = web3fsn.getAllSwaps(pageNo)


print('No. swaps = ',len(swap_dict),'\n')


for ii in range(len(swap_dict)):
    for key, value in swap_dict[ii].items():
        if key == 'timeStamp' or key == 'FromStartTime' or key == 'ToEndTime' or key␣
↪== 'Time' :
            value = web3fsn.numToDatetime(value)
        elif key == 'fromAsset' or key == 'toAsset':
            if value == web3fsn.tokens['FSN']:
                value = 'FSN'
        print(key, value)
```

Output from this code :-

```
>>>
No. swaps =  30

swapID 0x5a6cb08db87f0519471dcc9fb34a0a3e2163d6e1567db0c140f13e9dbeea51eb
timeStamp 2019-11-20 19:29:32+00:00
fromAddress 0x048c6f41542e55dd22a9a37b04b8122fa1ce1006
fromAsset FSN
toAsset FSN
recCreated 2019-11-20T19:29:53.000Z
height 947735
hash 0x8d3ba97b26a633d0e401ebb48546be109901100644144853bbcbaafe4b6020b9
size 10
Description
FromStartTime 1970-01-01 00:00:00+00:00
ToEndTime 2019-12-30 00:00:00+00:00
MinFromAmount 29100000000000000000
MinToAmount 2500000000000000000000
SwapSize 10
Targes []
Time 2019-11-20 19:29:19+00:00
ToAssetID 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

swapID 0x9bd0e524e4eef8c9585b43a6c7f6c293428212f8f1900c68dc33780dbe584958
timeStamp 2019-11-19 20:40:10+00:00
fromAddress 0x24714cc6408cf123979e37e03ed9dbcc84666620
fromAsset FSN
toAsset FSN
recCreated 2019-11-19T20:40:26.000Z
height 941439
hash 0x7428e50f375dcac87f661583f5e8c97dcb9c4e4adc90dc865fc748d4839aaf08
size 1
Description
FromStartTime 1970-01-01 00:00:00+00:00
ToEndTime 2019-11-30 00:00:00+00:00
MinFromAmount 25000000000000000000
```

(continues on next page)

(continued from previous page)

```
MinToAmount 500000000000000000000
SwapSize 1
Targes []
Time 2019-11-19 20:39:57+00:00
ToAssetID 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

swapID 0x2a79788a33f87b5a78de7805f6f8c361be47d16f9808b9fbc7f7fd7bf33644e6
timeStamp 2019-11-19 20:35:37+00:00
fromAddress 0x24714cc6408cf123979e37e03ed9dbcc84666620
fromAsset FSN
toAsset FSN
recCreated 2019-11-19T20:35:58.000Z
height 941418
hash 0x28740d793691c805c3611d813bcd48196853fed27aebb7fb230e5d0695ec4ce1
size 1
Description
FromStartTime 1970-01-01 00:00:00+00:00
ToEndTime 2019-12-31 00:00:00+00:00
MinFromAmount 50000000000000000000
MinToAmount 500000000000000000000
SwapSize 1
Targes []
Time 2019-11-19 20:35:11+00:00
ToAssetID 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

etc.
```

assetNameToAssetInfo()

2.7.3 assetNameToAssetInfo

def assetNameToAssetInfo(self, asset_name):

> “”” Retrieve information about a given verified asset name The asset *must be* ‘enabled’ and ‘whiteListEnabled’ in the fsnapi
>
> **Args:** asset_name (str) Short string asset idenfifier
>
> **Returns:** assetInfo (dict)
>
> “””

Here is an example of the function usage

```
from  web3fsnpy import Fsn

linkToChain = {
    'network'     : 'testnet',     # One of 'testnet', or 'mainnet'
    'provider'    : 'WebSocket',   # One of 'WebSocket', 'HTTP', or 'IPC'
    'gateway'     : 'default',
}

web3fsn = Fsn(linkToChain)
#
#
asset_name = 'FSN'
#
```

(continues on next page)

(continued from previous page)

```
#
assetInfo = web3fsn.assetNameToAssetInfo(asset_name)
#
print(assetInfo)
#
```

```
>>>assetInfo
{'assetID': '0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff',
'recCreated': '2019-07-09T06:40:19.000Z',
'recEdited': '2019-07-09T06:40:19.000Z',
'assetAuthority': '0xcf62374bc2b4e195ca7f2aecbe0076d9d4f89d1e',
'name': 'Fusion',
'shortName': 'FSN',
'image': 'EFSN_LIGHT.svg',
'erc20': 1,
'ethereum': 1,
'address': '0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff',
'disabled': 0,
'whiteListEnabled': 1,
'bitcoin': 0,
'decimals': 18,
'lockInDisabled': 1,
'reservedID': 1,
'totalFusionSupply': '0',
'msgSignedWithAssetAuthority':
↪'Signed:Fusion:0xffffffffffffffffffffffffffffffff
ffffffffffffffffffffffff:0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff:FSN
↪',
'msgSignature':
↪'0xf92792db22d1bb53c5c2bd8ccbdbfc1745c15fba35c1c0fb65d066f6dd03db937f5257
ec5244d04ce5584eb59029d11d453b9cf92057ecc194ce2d4f12bd97'
}
```

assetIdToAssetInfo()

2.7.4 assetIdToAssetInfo

def assetIdToAssetInfo(self, assetId): “”” Retrieve inforamtion about a given verified asset name The asset must be ‘enabled’ and ‘whiteListEnabled’ in the fsnapi

Args: assetId (hex str) Hex string asset idenfifier

Returns: assetInfo (dict) See *assetNameToAssetInfo* for typical output

“””

getAssetId()

2.7.5 getAssetId

def getAssetId(self, asset_name): “”” Retrieve the hexadecimal assetId for a given verified asset name The asset *must be* ‘enabled’ and ‘whiteListEnabled’ in the fsnapi

Args: asset_name (str) Short string asset idenfifier

Returns: assetId (hex str)

or None if not found or not enabled/whiteListEnabled

“””

Here is an example of the function usage

```
#
#
#asset_name = 'FSN'
asset_name = 'TST5'
blockNo = 'latest'
#
#
asset_Id = web3fsn.getAssetId(asset_name)
print('asset_Id = ',asset_Id)
#
if asset_Id != None:
    asset_dict = web3fsn.getAsset(asset_Id,blockNo)
#
#   print(asset_dict,'\n')
#
```

getAssetDecimals()

2.7.6 getAssetDecimals

def getAssetId(self, asset_name): “””Retrieve the decimals for a given verified asset name The asset must be **enabled** and **whiteListEnabled**

Args: asset_name (str) Short string asset idenfifier

Returns: decimals (int)

or None if not found or not enabled/whiteListEnabled

“””

fsnapiVerifiedAssetInfo()

2.7.7 fsnapiVerifiedAssetInfo

def fsnapiVerifiedAssetInfo(self): “””Get a list of the shortnames of all assets that are **whitelisted** and **verified**

Args: None

Returns: verifiedAssets (list) e.g. [‘BTC’,’ETH’,’FSN’,. . .]

“””

fsnapi_swaps_pubkey()

2.7.8 fsnapi_swaps_pubkey

def fsnapi_swaps_pubkey(self, account, pageNo): “””Output a list of all swaps have been generated by a public pub_key

Args: account (hex string) public key, PageNo (int) The data is served with 100 records per page, starting at page 0. Simply increment until the list is exhausted and the length of the output is less than 100.

Returns: swap_dict (dict)

“””

fsnapi_swaps_target()

2.7.9 fsnapi_swaps_target

def fsnapi_swaps_target(self, account, pageNo): “””Output a list of all swaps have been targetted at a public pub_key

Args: account (hex string) public key, PageNo (int) The data is served with 100 records per page, starting at page 0. Simply increment until the list is exhausted and the length of the output is less than 100.

Returns: swap_dict (dict)

“””

transactionNoTicketsDesc()

2.7.10 transactionNoTicketsDesc

def transactionNoTicketsDesc(self, pageNo): “””Output transactions from the blockchain with the most recent first and ignoring ticket purchase transactions

Args: PageNo (int) The data is served with 100 records per page, starting at page 0. Simply increment until the list is exhausted and the length of the output is less than 100.

Returns: Txs (dict)

“””

takeSwapsDesc()

2.7.11 takeSwapsDesc

def takeSwapsDesc(self, pageNo): “””Output a list of all takeSwap transactions, wiht the most recent first

Args: PageNo (int) The data is served with 100 records per page, starting at page 0. Simply increment until the list is exhausted and the length of the output is less than 100.

Returns: swaps (dict)

“””

2.8 Miscellaneous

numToDatetime()

2.8.1 numToDatetime

def numToDatetime(self, tdelta): “””Converts the simple integer number of seconds since 1970/01/01:0000 UTC to a timezone enabled python DateTime object

Args: tdelta (int),

Returns: dateTime (DateTime object)

“””

datetimeToHex()

2.8.2 datetimeToHex

def datetimeToHex(self, dateTime): “””Converts a python timezone enabled DateTime object to a hex string representing the number of seconds since 1970/01/01:0000 UTC

Args: dateTime (DateTime object)

Returns: dtHex (hex string)

“””

datetimeToInt()

2.8.3 datetimeToInt

def datetimeToInt(self, dateTime): “””Converts a python timezone enabled DateTime object to an int representing the number of seconds since 1970/01/01:0000 UTC

Args: dateTime (DateTime object)

Returns: dt (int)

“””

hex2a()

2.8.4 hex2a

def hex2a(self, datastr): “””Decodes the ‘data’ string in a Fusion blockchain transaction to reveal the ‘to’ hexadecimal address of the recipient

Args: datastr (string)

Returns: pub_key (20 char hex string)

“””

CHAPTER 3

Code Examples

Snippets from these example are used in the *Functions in the Fsn Class* descriptions.

fsnBlockheight.py Basic example showing how to get the current block height.

fsnGetBalance.py Simple example to get the FSN balance of an account.

fsnBuyTicket.py Buying a ticket, checking to see if autobuy ticket is on.

fsnAllTickets.py Listing all tickets and their block heights on the blockchain and showing the totalNumberOfTickets function. Also shows usage of the ticketPrice and getStakeInfo functions.

fsnTicketsByAddress.py Listing tickets for an account, including their start and end times.

fsnSendRawFSN.py Send FSN with a raw transaction and show getTransactionCount and getTransaction.

fsnGetAsset.py Show how to get asset information from the fsnapi.

fsnAssetNameToAssetInfo.py Get asset information, given the asset name (only for whitelisted and enabled assets - 'verified').

fsnGetAsset.py Get asset information about any asset on the blockchain, given its asset ID or asset name (shows getAssetId function).

fsnCreateRawAsset.py Show how to create an asset.

fsnInc_and_DecRawAsset.py Show how to increase or decrease the supply of an asset (if it can be changed).

fsnSendRawAsset.py Demonstrate how to send an asset on the blockchain to another wallet.

fsnGetAllTimelockInfo.py Get all the timelocks for all assets from an account.

fsnGetTimelockInfo.py Get timelock inforamtion about one asset ID.

fsnToAndFromRawTimeLock.py Sending assets to and from timelock.

fsnSendToRawTimeLock.py Sending assets to another address as timelock.

fsnTimeLockToRawTimeLock.py Change the timelock on an asset.

fsnGetAllSwaps.py Get information on all swaps in the Quantum Swap Marketplace.

fsnMakeAndRecallRawSwap.py Create a new swap and then recall it.

fsnTakeRawSwap.py Take all or part of a swap. Example also shows getSwap.

fsnGenNotation.py Generate a new USAN for an account and then show how to get the public key for it (and vice-versa).

fsnOfflineTransactions.py Show how to separate out the preparation of transactions from actaully signing and transmitting them.

CHAPTER 4

Offline Transactions

We can separate out the preparation of a transaction (sending FSN, or assets, creating Notations, timelocks, making swaps etc.) from the actual signing and sending of the transaction onto the blockchain. In this way it is possible for a different group of people to be responsible for the organising the transactions to those who have control of the wallets themselves, perhaps using more secure computers.

Here we show an example of how this is accomplished. The initial part is done using the optional *prepareOnly* flag in the function calls. The transaction dictionaries are written as JSON strings to a text file. This same text file could be sent to someone else and used to sign and send the transactions at a later time.

```
asset_dict = web3fsn.getAsset(asset_Id,'latest')
#print(asset_dict)
print('The asset has the symbol ',asset_dict['Symbol'],' and decimals ',asset_dict[
↪'Decimals'])

nonce = web3fsn.getTransactionCount(pub_key_sender)  # Get the nonce for the wallet

# Construct the transaction

transaction = {
  'from':       pub_key_sender,
  'to':         pub_key_receiver,
  'nonce':      nonce,
  'asset':      asset_Id,
  'value':      int(number_to_transfer*10**float(asset_dict['Decimals'])),    # This␣
↪is the integer number of the smallest unit that can be sent, defined by the␣
↪decimals of the asset.
}

# Open a file to store json data
fp = open('fusion_transaction_dict.json','w')

Tx_dict = web3fsn.sendRawAsset(transaction, prepareOnly=True)     # Set the␣
↪prepareOnly flag to stop them being signed now.
#
```

(continues on next page)

(continued from previous page)

```
# Store the data
json_tx = json.dumps(Tx_dict)
fp.write(json_tx+'\n')
#
# Add another transaction
transaction['nonce'] = nonce + 1
Tx_dict = web3fsn.sendRawAsset(transaction, prepareOnly=True)
json_tx = json.dumps(Tx_dict)
fp.write(json_tx+'\n')
#
# Send some FSN too
value = web3fsn.toWei(0.02,'ether')     # How much FSN are we sending?

transaction = {
            "from"  : pub_key_sender,
            "to"    : pub_key_receiver,
            "nonce" : nonce + 2,
            "value" : value,
}

# Send the raw transaction.
Tx_dict = web3fsn.sendRawTransaction(transaction, prepareOnly=True)
json_tx = json.dumps(Tx_dict)
fp.write(json_tx+'\n')
#
#
fp.close()
del web3fsn     # Clean up and delete the Fsn object
#
#
# At a later time, or from a more secure computer, you can sign and transmit the data
#
print('\nWe have finished generating the transactions now. At a later time we can␣
↪actually sign and transmit them...\n')
#
#   Remember to set your environment variables to run this test



linkToChain = {
    'network'     : 'testnet',                          # One of 'testnet', or␣
↪'mainnet'
    'provider'    : 'HTTP',                             # One of 'WebSocket', 'HTTP',␣
↪or 'IPC'
    'gateway'     : 'default',                          # Either set to 'default', or␣
↪specify your uri endpoint
    'private_key'     : os.environ["FSN_PRIVATE_KEY"],  # This time we need a private␣
↪key
}
#
web3fsn = Fsn(linkToChain)
#
# Get the transaction data we stored previously
fp = open('fusion_transaction_dict.json','r')
#
ii = 1
#
```

(continues on next page)

(continued from previous page)

```
for line in fp:
    Tx_dict = json.loads(line)
    print('Signing and transmitting ',ii)
    TxHash = web3fsn.signAndTransmit(Tx_dict)
    print('Transaction hash = ',TxHash)
    web3fsn.waitForTransactionReceipt(TxHash, timeout=20)
    ii = ii+1
#
print('\nFinished')
#
os.remove('fusion_transaction_dict.json')
fp.close()
#
```

CHAPTER 5

For Help and to Join the Community

Contact Marcel Curé @marcelsecu on the Fusion Developers Community Channel https://t.me/FsnDevCommunity

You can easily interact with other Fusion developers through its Developer's Telegram channel and through the Fusion Open Source Community (FOSC). Here you can discuss new project ideas, or seek technical assistance from other developers and the Fusion technical team.

If you have some code that you would like to add to the repository, please create a pull request to https://github.com/FUSIONFoundation/web3fsnpy

and let's create a powerful resource for all developers.

Don't be a stranger!

CHAPTER 6

License

CHAPTER 7

Search

- search

Index

T

W