ScanWatch

Release 0.2.0

EtWnn

Jan 05, 2022

CONTENTS

CHAPTER

ONE

NOTE

This library is developed and maintained by EtWnn, feel free to drop your suggestions or remarks in the discussion tab. You are also welcome to contribute by submitting PRs.

Source Code: https://github.com/EtWnn/ScanWatch

Documentation: https://scanwatch.readthedocs.io

This library is a local tracker of transactions for the Ethereum chain, the Binance Smart chain and the Polygon chain. It is a simple single-point interface with the etherscan, bscscan and polygonscan APIs.
This library will save locally the transactions to gain time and avoid over-calling the APIs.

CHAPTER

TWO

ANNOUNCEMENT

If you previously used this library with a version inferior to 0.1.3, please head here to correct a potential bug in the database.

CHAPTER

THREE

QUICK TOUR

3.1 1. API Keys

You will need to generate API tokens to use this library:

1. Ethereum chain: go on etherscan
2. Binance Smart chain: go on bscscan
3. Polygon chain: go on polygonscan

(If you want to use several chains, you will need an API token for each).

3.2 2. Installation

`ScanWatch` is available on PYPI, install with `pip`:

```
pip install ScanWatch
```

You can also install the latest developments (not stable):

```
pip install git+https://github.com/EtWnn/ScanWatch.git@develop
```

3.3 3. Manager

The manager is the object that you will use to update and get the transactions.
It is instantiated with an API token and an address.

Example for Ethereum:

```
from ScanWatch.ScanManager import ScanManager
from ScanWatch.utils.enums import NETWORK

api_token = "<ETH_API_TOKEN>"
address = "<YOUR_ETH_ADDRESS>"

manager = ScanManager(address, NETWORK.ETHER, api_token)
```

Example for BSC:

```
from ScanWatch.ScanManager import ScanManager
from ScanWatch.utils.enums import NETWORK

api_token = "<BSC_API_TOKEN>"
address = "<YOUR_BSC_ADDRESS>"

manager = ScanManager(address, NETWORK.BSC, api_token)
```

Example for Polygon:

```
from ScanWatch.ScanManager import ScanManager
from ScanWatch.utils.enums import NETWORK

api_token = "<POLYGON_API_TOKEN>"
address = "<YOUR_POLYGON_ADDRESS>"

manager = ScanManager(address, NETWORK.POLYGON, api_token)
```

3.4 4. Transactions Update

Once the manager is setup, you can update the locally saved transactions:

```
manager.update_all_transactions()
# all transactions updated for address 0xaAC...748E8: 100%|| 4/4 [00:02<00:00,  1.86it/s]
```

This needs to be done only when new transactions have been made since the last time you called the update method.

3.5 5. Transactions

To fetch the transactions that have been previously saved, just use the methods below. (see the documentation for more details).

```
from ScanWatch.utils.enums import TRANSACTION

manager.get_transactions(TRANSACTION.NORMAL)  # normal transactions

manager.get_transactions(TRANSACTION.ERC20)  # erc20 transactions

manager.get_transactions(TRANSACTION.ERC721)  # erc721 transactions

manager.get_transactions(TRANSACTION.INTERNAL)  # internal transactions
```

3.6 6. Holdings

The manager can also give you the current tokens hold by an address:

For erc20 tokens:

```
manager.get_erc20_holdings()
```

```
{
    'USDC': Decimal('50'),
    'AllianceBlock Token': Decimal('12458.494516884'),
    'Blockchain Certified Data Token': Decimal('75174'),
    'Compound': Decimal('784.24998156'),
    'ZRX': Decimal('3.1')
}
```

For erc721 tokens:

```
manager.get_erc721_holdings()
```

```
[
    {
        'contractAddress': '0x8azd48c9ze46azx1e984fraz4da9zz8dssad49ct',
        'tokenID': '78941',
        'count': 1,
        'tokenName': 'SUPER NFT GAME',
        'tokenSymbol': 'Hero'
    },
    {
        'contractAddress': '0x6edd39bdba2fazs3db5fxd86908789cbd905f04d',
        'tokenID': '33001',
        'count': 1,
        'tokenName': 'MY FAV NFT ARTIST HANDMADE THIS',
        'tokenSymbol': 'dubious thing'
    }
]
```

CHAPTER

FOUR

MAIN / TEST NETS

If you want to switch from main to test nets, you can specify the net name at the manager creation:

```
manager = ScanManager(address, <network>, api_token, <net_name>)
```

Supported nets are:

- For Ethereum: “main”, “goerli”, “kovan”, “rinkeby”, “ropsten”
- For BSC: “main”, “test”
- For Polygon: “main”, “test”

CHAPTER

FIVE

DONATION

If this library has helped you in any way, feel free to help me
With your donation, I will be able to keep working on this project and add new features. Thank you!

- **BTC**: 14ou4fMYoMVYbWEKnhADPJUNVytWQWx9HG
- **ETH**, **BSC**, **Polygon**: 0xA20be1f02B1C9D4FF1442a0F0e7c089fcDd59407
- **LTC**: LfHgc969RFUjnmyLn41SRDvmT146jUg9tE
- **EGLD**: erd1qk98xm2hgztvmq6s4jwtk06g6laattewp6vh20z393drzy5zzfrq0gaefh

5.1 ScanManager

class ScanWatch.ScanManager.**ScanManager**(*address: str*, *nt_type:* ScanWatch.utils.enums.NETWORK, *api_token: str*, *net: str = 'main'*)

This class is the interface between the user, the API and the Database

__init__(*address: str*, *nt_type:* ScanWatch.utils.enums.NETWORK, *api_token: str*, *net: str = 'main'*)

Initiate the manager

Parameters

- **address** (*str*) – address to monitor
- **nt_type** (NETWORK) – type of the network
- **api_token** (*str*) – token to communicate with the API
- **net** (*str, default 'main'*) – name of the network, used to differentiate main and test nets

get_erc20_holdings() → Dict

Return the amount of every erc20 the address holds at the last update time. WARNING: Some tokens trigger non-erc20 events, such as internal exchange fee. This will not be picked up by this function. As a consequence, the balance of such tokens might be wrong.

Returns a dictionary of token amount per token name

Return type Dict

get_erc721_holdings() → List[Dict]

Return the erc721 tokens that the address holds at the time of the last update

Returns List of erc721 tokens owned by the address

Return type List[Dict]

get_transactions(*tr_type:* ScanWatch.utils.enums.TRANSACTION)
Return the transactions of the provided type that are saved locally for the address of the manager

Parameters **tr_type** (TRANSACTION) – type of transaction to fetch

Returns list of transactions

Return type List[Dict]

update_all_transactions()
Update all the transactions for the address

Returns None

Return type None

update_transactions(*tr_type:* ScanWatch.utils.enums.TRANSACTION)
Update the transactions of a certain type in the database

Parameters **tr_type** (TRANSACTION) – type of transaction to update

Returns None

Return type None

5.2 Client

class ScanWatch.Client.**Client**(*api_token: str*, *nt_type:* ScanWatch.utils.enums.NETWORK, *net: str = 'main'*)
Client the API: https://etherscan.io/apis https://bscscan.com/apis https://polygonscan.com/apis

BASE_URLS = {<NETWORK.BSC: 2>: {'main': 'https://api.bscscan.com/api', 'test': 'https://api-testnet.bscscan.com/api'}, <NETWORK.ETHER: 1>: {'main': 'https://api.etherscan.io/api', 'goerli': 'https://api-goerli.etherscan.io/api', 'kovan': 'https://api-kovan.etherscan.io/api', 'rinkeby': 'https://api-rinkeby.etherscan.io/api', 'ropsten': 'https://api-ropsten.etherscan.io/api'}, <NETWORK.POLYGON: 3>: {'main': 'https://api.polygonscan.com/api', 'test': 'https://api-testnet.polygonscan.com/api'}}

__init__(*api_token: str*, *nt_type:* ScanWatch.utils.enums.NETWORK, *net: str = 'main'*)

Parameters

- **api_token** (*str*) – token for the api
- **nt_type** (NETWORK) – type of the network
- **net** (*str, default 'main'*) – name of the network, used to differentiate main and test nets

get_balance(*address: str*) → float
fetch the current balance of an address

Parameters **address** (*str*) – address

Returns ETH amount

Return type float

get_erc20_transactions(*address: str*, *start_block: Optional[int] = None*, *end_block: Optional[int] = None*)

fetch erc20 transactions on an address

Parameters

- **address** (*str*) – address
- **start_block** (*Optional[int]*) – fetch transactions starting with this block
- **end_block** (*Optional[int]*) – fetch transactions until this block

Returns List of transactions

Return type List[Dict]

```
[{'blockNumber': '108941',
  'timeStamp': '148216518',
  'hash': '0948461czecc9ze8e4vsvbq94sd96',
  'nonce': '23908745',
  'blockHash': '0x74a984dz56c13v8ze9q451vda',
  'from': 'zazd9f1wsqda84zds5qd6zda',
  'contractAddress': 'azd984f1edazdadqfefa',
  'to': '84cazd984csgzefa984zq5s1c',
  'value': '1524896000000000000',
  'tokenName': 'ChainLink Token',
  'tokenSymbol': 'LINK',
  'tokenDecimal': '18',
  'transactionIndex': '45',
  'gas': '200001',
  'gasPrice': '100000000000',
  'gasUsed': '51481',
  'cumulativeGasUsed': '1491504',
  'input': 'deprecated',
  'confirmations': '1948513'},
...
]
```

get_erc721_transactions(*address: str*, *start_block: Optional[int] = None*, *end_block: Optional[int] = None*)

fetch erc721 transactions on an address

Parameters

- **address** (*str*) – address
- **start_block** (*Optional[int]*) – fetch transactions starting with this block
- **end_block** (*Optional[int]*) – fetch transactions until this block

Returns List of transactions

Return type List[Dict]

get_internal_transactions(*address: str*, *start_block: Optional[int] = None*, *end_block: Optional[int] = None*)

fetch internal transactions on an address

Parameters

- **address** (*str*) – address

- **start_block** (*Optional[int]*) – fetch transactions starting with this block
- **end_block** (*Optional[int]*) – fetch transactions until this block

Returns List of transactions

Return type List[Dict]

get_mined_blocks(*address: str*, *start_block: Optional[int] = None*, *end_block: Optional[int] = None*)
fetch mined blocks by an address

Parameters

- **address** (*str*) – network address
- **start_block** (*Optional[int]*) – fetch mined blocks starting with this block
- **end_block** (*Optional[int]*) – fetch mined blocks until this block

Returns List of mined blocks

Return type List[Dict]

get_normal_transactions(*address: str*, *start_block: Optional[int] = None*, *end_block: Optional[int] = None*)
fetch normal transactions on an address

Parameters

- **address** (*str*) – address
- **start_block** (*Optional[int]*) – fetch transactions starting with this block
- **end_block** (*Optional[int]*) – fetch transactions until this block

Returns List of transactions

Return type List[Dict]

```
[{'blockNumber': '10272495',
  'timeStamp': '1485965131',
  'hash': 'd9azfv1q9zf84zr15f49f49zef1z3sd1g98t1b6',
  'nonce': '9846651',
  'blockHash': 'zad94v16s1qef9a4f9v1r53b1sq64daf',
  'from': '496a1ef65s1dbv4a96z513svez965q',
  'contractAddress': 'az41f8ze1f63q5s1gv89ez49',
  'to': 'az49f84161ac89s4ef984a96e',
  'value': '1785400000000000000',
  'tokenName': 'ChainLink Token',
  'tokenSymbol': 'LINK',
  'tokenDecimal': '18',
  'transactionIndex': '79',
  'gas': '200001',
  'gasPrice': '100000000000',
  'gasUsed': '84215',
  'cumulativeGasUsed': '1531404',
  'input': 'deprecated',
  'confirmations': '119452'},
...
]
```

static get_result(*url: str*)
call the API with an url, raise if the status is not ok and return the API result

Parameters **url** (*str*) – url to request

Returns API result

Return type depend of the endpoint

get_url_request(***kwargs*) → str
Construct the url to make a request to the API

Parameters **kwargs** (*Any*) – keywords args for the endpoint

Returns

Return type

5.3 DataBase

class ScanWatch.storage.ScanDataBase.**ScanDataBase**(*name: str = 'scan_db'*)
Handles the recording of the address transactions in a local database

__init__(*name: str = 'scan_db'*)
Initialise a Scan database instance

Parameters **name** (*str*) – name of the database

add_transactions(*address: str*, *nt_type:* ScanWatch.utils.enums.NETWORK, *net: str*, *tr_type:* ScanWatch.utils.enums.TRANSACTION, *transactions: List[Dict]*)
Add a list of transactions to the database

Parameters

- **address** (*str*) – address involved in the transaction
- **nt_type** (NETWORK) – type of network
- **net** (*str*) – name of the network, used to differentiate main and test nets
- **tr_type** (TRANSACTION) – type of the transaction to record
- **transactions** (*List[Dict]*) – list of the transaction to record

Returns None

Return type None

get_last_block_number(*address: str*, *nt_type:* ScanWatch.utils.enums.NETWORK, *net: str*, *tr_type:* ScanWatch.utils.enums.TRANSACTION) → int
Return the last block number seen in recorded transactions (per address, type of transaction and network)
If None are found, return 0

Parameters

- **address** (*str*) – address involved in the transactions
- **nt_type** (NETWORK) – type of network
- **net** (*str*) – name of the network, used to differentiate main and test nets
- **tr_type** (TRANSACTION) – type of the transaction to fetch

Returns last block number

Return type int

get_transactions(*address: str*, *nt_type:* ScanWatch.utils.enums.NETWORK, *net: str*, *tr_type:* ScanWatch.utils.enums.TRANSACTION) → List[Dict]

Return the List of the transactions recorded in the database

Parameters

- **address** (*str*) – address involved in the transactions
- **nt_type** (NETWORK) – type of network
- **net** (*str*) – name of the network, used to differentiate main and test nets
- **tr_type** (TRANSACTION) – type of the transaction to fetch

Returns list of the transaction recorded

Return type List[Dict]

class ScanWatch.storage.DataBase.**DataBase**(*name: str*)

This class will be used to interact with sqlite3 databases without having to generates sqlite commands

__init__(*name: str*)

Initialise a DataBase instance

Parameters **name** (*str*) – name of the database

add_row(*table: ScanWatch.storage.tables.Table*, *row: Tuple*, *auto_commit: bool = True*, *update_if_exists: bool = False*)

Add a row to a table

Parameters

- **table** (*Table*) – table to add a row to
- **row** (*Tuple*) – values to add to the database
- **auto_commit** (*bool*) – if the database state should be saved after the changes
- **update_if_exists** (*bool*) – if an integrity error is raised and this parameter is true, will update the existing row

Returns None

Return type None

add_rows(*table: ScanWatch.storage.tables.Table*, *rows: List[Tuple]*, *auto_commit: bool = True*, *update_if_exists: bool = False*)

Add several rows to a table

Parameters

- **table** (*Table*) – table to add a row to
- **rows** (*List[Tuple]*) – list of values to add to the database
- **auto_commit** (*bool*) – if the database state should be saved after the changes
- **update_if_exists** (*bool*) – if an integrity error is raised and this parameter is true, will update the existing row

Returns None

Return type None

close()

Close the connection with the sqlite3 database

Returns None

Return type None

commit()
Submit and save the database state

Returns None

Return type None

connect()
Connect to the sqlite3 database

Returns None

Return type None

create_table(*table: ScanWatch.storage.tables.Table*)
Create a table in the database

Parameters **table** (*Table*) – Table instance with the config of the table to create

Returns None

Return type None

drop_all_tables()
Drop all the tables existing in the database

Returns None

Return type None

drop_table(*table: Union[ScanWatch.storage.tables.Table, str]*)
Delete a table from the database

Parameters **table** (*Union[Table, str]*) – table or table name to drop

Returns None

Return type None

get_all_rows(*table: ScanWatch.storage.tables.Table*) → List[Tuple]
Get all the rows of a table

Parameters **table** (*Table*) – table to get the rows from

Returns all the rows of the table

Return type List[Tuple]

get_all_tables() → List[Tuple]
Return all the tables existing in the database

Returns tables descriptions

Return type List[Tuple]

get_conditions_rows(*table: ScanWatch.storage.tables.Table*, *selection: Union[str, List[str]] = '*'*, *conditions_list: Optional[List[Tuple[str, ScanWatch.storage.DataBase.SQLConditionEnum, Any]]] = None*, *order_list: Optional[List[str]] = None*) → List[Tuple]
Select rows with optional conditions and optional order

Parameters

- **table** (*Table*) – table to select the rows from
- **selection** (*Union[str, List[str]]*) – list of column or SQL type selection

- **conditions_list** (*Optional[List[Tuple[str, SQLConditionEnum, Any]]]*) – list of conditions to select the row
- **order_list** (*Optional[List[str]]*) – List of SQL type order by

Returns the selected rows

Return type List[Tuple]

static get_create_cmd(*table: ScanWatch.storage.tables.Table*) → str
Return the command in string format to create a table in the database

Parameters **table** (*Table*) – Table instance with the config if the table to create

Returns execution command for the table creation

Return type str

get_row_by_key(*table: ScanWatch.storage.tables.Table*, *key_value*) → Optional[Tuple]
Get the row identified by a primary key value from a table

Parameters

- **table** (*Table*) – table to fetch the key from
- **key_value** (*Any*) – value of the primary key

Returns the raw row of of the table

Return type Optional[Tuple]

update_row(*table: ScanWatch.storage.tables.Table*, *row: Tuple*, *auto_commit=True*)
Update the value of a row in a table

Parameters

- **table** (*Table*) – table to get updated
- **row** (*Tuple*) – values to update
- **auto_commit** (*bool*) – if the database state should be saved after the changes

Returns None

Return type None

class ScanWatch.storage.DataBase.**SQLConditionEnum**(*value*)
Enumeration for SQL comparison operator

https://www.techonthenet.com/sqlite/comparison_operators.php

diff = '!='

equal = '='

greater = '>'

greater_equal = '>='

lower = '<'

lower_equal = '<='

5.4 Enums

class ScanWatch.utils.enums.**NETWORK**(*value*)
An enumeration.

BSC = 2

ETHER = 1

POLYGON = 3

class ScanWatch.utils.enums.**TRANSACTION**(*value*)
An enumeration.

ERC20 = 3

ERC721 = 4

INTERNAL = 2

NORMAL = 1

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

D

E

G