py_crypto_hd_wallet

Emanuele Bellocchia

Apr 24, 2024

CONTENTS

CHAPTER

ONE

INTRODUCTION

This package contains a very basic implementation of a HD (Hierarchical Deterministic) wallet based on my bip_utils library.It is basically a nice wrapper for the *bip_utils* library for generating mnemonics, seeds, public/private keys and addresses. Therefore, it has no network functionalities.The supported coins are the same of the bip_utils library, so check the related page.

CHAPTER

TWO

INSTALL THE PACKAGE

The package requires Python 3, it is not compatible with Python 2. To install it:

- Using *pip*, from this directory (local):

```
pip install .
```

- Using *pip*, from PyPI:

```
pip install py_crypto_hd_wallet
```

NOTE: if you are using an Apple M1, please make sure to update *coincurve* (required by *bip_utils*) to version 17.0.0 otherwise it won't work.

CHAPTER

THREE

TEST AND COVERAGE

Install develop dependencies:

```
pip install -r requirements-dev.txt
```

To run tests:

```
python -m unittest discover
```

To run tests with coverage:

```
coverage run -m unittest discover
coverage report
```

To run code analysis, just execute the `analyze_code` script.

CHAPTER

FOUR

MODULES DESCRIPTION

- BIP wallet
- Algorand wallet
- Cardano Shelley wallet
- Electrum V1 wallet
- Electrum V2 wallet
- Monero wallet
- Substrate wallet

CHAPTER

FIVE

EXAMPLES OF WALLET JSON OUTPUTS

CHAPTER

SIX

DOCUMENTATION

The library documentation is available at py-crypto-hd-wallet.readthedocs.io.

CHAPTER

SEVEN

BUY ME A COFFEE

You know, I'm italian and I love drinking coffee (especially while coding :D). So, if you'd like to buy me one:

- BTC: `bc1qq4r9cglwzd6f2hzxvdkucmdejvr9h8me5hy0k8`
- ERC20/BEP20: `0xf84e4898E5E10bf1fBe9ffA3EEC845e82e364b5B`

Thank you very much for your support.

CHAPTER

EIGHT

LICENSE

This software is available under the MIT license.

CHAPTER

NINE

MODULES

9.1 py_crypto_hd_wallet

9.1.1 algorand

9.1.1.1 hd_wallet_algorand

Module for generating Algorand wallets.

class HdWalletAlgorand(*wallet_name: str*, *bip_obj: Bip44Base*, *mnemonic: str = ''*, *seed_bytes: bytes = b''*)

Bases: *HdWalletBase*

HD wallet Algorand class. It allows to generate an Algorand wallet like the official one.

m_bip_obj: Bip44Base

Generate(***kwargs: Any*) → None

Generate wallet keys and addresses.

IsWatchOnly() → bool

Get if the wallet is watch-only.

Returns :

bool: True if watch-only, false otherwise

9.1.1.2 hd_wallet_algorand_enum

Module with enums for Algorand wallets.

class HdWalletAlgorandDataTypes(*value*)

Bases: *HdWalletDataTypes*

Enumerative for wallet Algorand data types.

WALLET_NAME = 1

COIN_NAME = 2

MNEMONIC = 3

SEED_BYTES = 4

KEY = 5

class HdWalletAlgorandKeyTypes(*value*)

Bases: *HdWalletKeyTypes*

Enumerative for wallet Algorand key types.

PRIV = 1

PUB = 2

ADDRESS = 3

9.1.1.3 hd_wallet_algorand_factory

Module for creating Algorand wallet factories.

class HdWalletAlgorandFactory

Bases: `object`

HD wallet Algorand factory class. It allows a HdWalletAlgorand to be created in different way.

CreateRandom(*wallet_name: str*, *words_num: AlgorandWordsNum = AlgorandWordsNum.WORDS_NUM_25*, *lang: AlgorandLanguages = AlgorandLanguages.ENGLISH*) → *HdWalletBase*

Create wallet randomly.

Parameters

- **wallet_name** (*str*) – Wallet name
- **words_num** (*HdWalletAlgorandWordsNum, optional*) – Words number (default: 25)
- **lang** (*HdWalletAlgorandLanguages, optional*) – Language (default: English)

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
TypeError – If words number is not a HdWalletAlgorandWordsNum enum or language is not a HdWalletAlgorandLanguages enum

CreateFromMnemonic(*wallet_name: str*, *mnemonic: str*) → *HdWalletBase*

Create wallet from mnemonic.

Parameters

- **wallet_name** (*str*) – Wallet name
- **mnemonic** (*str*) – Mnemonic

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the mnemonic is not valid

CreateFromSeed(*wallet_name: str*, *seed_bytes: bytes*) → *HdWalletBase*

Create wallet from seed.

Parameters

- **wallet_name** (*str*) – Wallet name
- **seed_bytes** (*bytes*) – Seed bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the seed is not valid

CreateFromPrivateKey(*wallet_name: str*, *priv_key_bytes: bytes*) → *HdWalletBase*

Create wallet from private key.

Parameters

- **wallet_name** (*str*) – Wallet name
- **priv_key_bytes** (*bytes*) – Private key bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the private key is not valid

CreateFromPublicKey(*wallet_name: str*, *pub_key_bytes: bytes*) → *HdWalletBase*

Create wallet from public key.

Parameters

- **wallet_name** (*str*) – Wallet name
- **pub_key_bytes** (*bytes*) – Public key bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the public key is not valid

9.1.1.4 hd_wallet_algorand_keys

Module with helper class for storing Algorand keys.

class HdWalletAlgorandKeys(*bip_obj: Bip44Base*)

Bases: *HdWalletKeysBase*

HD wallet Algorand keys class. It creates keys from an Algorand object and stores them. Keys can be got individually, as dictionary or in JSON format.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

9.1.2 bip

9.1.2.1 hd_wallet_bip

Module for generating wallets based on BIP specifications.

class HdWalletBip(*wallet_name: str*, *bip_obj: Bip44Base*, *mnemonic: str = ''*, *passphrase: str = ''*, *seed_bytes: bytes = b''*)

Bases: *HdWalletBase*

HD wallet BIP class. It allows to generate a complete wallet based on BIP specifications.

m_bip_obj: Bip44Base

Generate(***kwargs: Any*) → None

Generate wallet keys and addresses.

Parameters

- **acc_idx** (*int, optional*) – Account index (default: 0)
- **change_idx** (*HdWalletBipChanges, optional*) – Change index (default: external)
- **addr_num** (*int, optional*) – Number of addresses to be generated (default: 20)
- **addr_off** (*int, optional*) – Starting address index (default: 0)

IsWatchOnly() → bool

Get if the wallet is watch-only.

Returns :
bool: True if watch-only, false otherwise

9.1.2.2 hd_wallet_bip_addr

Module with helper class for storing BIP addresses.

class HdWalletBipAddresses(*bip_obj: Bip44Base*, *addr_num: int*, *addr_off: int*)

Bases: *HdWalletAddrBase*

HD wallet BIP addresses class. It creates addresses from a Bip object and stores them. Addresses can be got individually, as dictionary or in JSON format.

m_addr_off: int

m_addr: List[Any]

m_dict_key_str_format: str

9.1.2.3 hd_wallet_bip_enum

Module with enums for BIP wallets.

class HdWalletBipDataTypes(*value*)

Bases: *HdWalletDataTypes*

Enumerative for wallet BIP data types.

WALLET_NAME = 1

COIN_NAME = 2

SPEC_NAME = 3

MNEMONIC = 4

PASSPHRASE = 5

SEED_BYTES = 6

ACCOUNT_IDX = 7

CHANGE_IDX = 8

MASTER_KEY = 9

PURPOSE_KEY = 10

COIN_KEY = 11

ACCOUNT_KEY = 12

CHANGE_KEY = 13

ADDRESS_OFF = 14

ADDRESS = 15

class HdWalletBipKeyTypes(*value*)

Bases: *HdWalletKeyTypes*

Enumerative for wallet BIP key types.

EX_PRIV = 1

RAW_PRIV = 2

WIF_PRIV = 3

EX_PUB = 4

RAW_COMPR_PUB = 5

RAW_UNCOMPR_PUB = 6

ADDRESS = 7

9.1.2.4 hd_wallet_bip_factory

Module for creating BIP wallet factories.

class HdWalletBipFactoryConst

Bases: object

Class container for HD wallet BIP factory constants.

BIP_COIN_TO_CLASS: Dict[Type[BipCoins], Type[Bip44Base]] = {<enum 'Bip44Coins'>: <class 'bip_utils.bip.bip44.bip44.Bip44'>, <enum 'Bip49Coins'>: <class 'bip_utils.bip.bip49.bip49.Bip49'>, <enum 'Bip84Coins'>: <class 'bip_utils.bip.bip84.bip84.Bip84'>, <enum 'Bip86Coins'>: <class 'bip_utils.bip.bip86.bip86.Bip86'>}

class HdWalletBipFactory(*coin_type: BipCoins*)

Bases: object

HD wallet BIP factory class. It allows a HdWalletBip to be created in different ways.

m_bip_cls: Type[Bip44Base]

m_bip_coin: BipCoins

CreateRandom(*wallet_name: str*, *words_num: Bip39WordsNum = Bip39WordsNum.WORDS_NUM_24*, *lang: Bip39Languages = Bip39Languages.ENGLISH*) → *HdWalletBase*

Create wallet randomly.

Parameters

- **wallet_name** (*str*) – Wallet name
- **words_num** (*HdWalletBipWordsNum, optional*) – Words number (default: 24)
- **lang** (*HdWalletBipLanguages, optional*) – Language (default: English)

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
TypeError – If words number is not a HdWalletBipWordsNum enum or language is not a HdWalletBipLanguages enum

CreateFromMnemonic(*wallet_name: str*, *mnemonic: str*, *passphrase: str = ''*) → *HdWalletBase*

Create wallet from mnemonic.

Parameters

- **wallet_name** (*str*) – Wallet name
- **mnemonic** (*str*) – Mnemonic
- **passphrase** (*str, optional*) – Passphrase for protecting mnemonic, empty if not specified

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the mnemonic is not valid

CreateFromSeed(*wallet_name: str*, *seed_bytes: bytes*) → *HdWalletBase*

Create wallet from seed.

Parameters

- **wallet_name** (*str*) – Wallet name
- **seed_bytes** (*bytes*) – Seed bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the seed is not valid

CreateFromExtendedKey(*wallet_name: str*, *ex_key_str: str*) → *HdWalletBase*

Create wallet from extended key.

Parameters

- **wallet_name** (*str*) – Wallet name
- **ex_key_str** (*str*) – Extended key string

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the extended key is not valid

9.1.2.5 hd_wallet_bip_keys

Module with helper class for storing BIP keys.

class HdWalletBipKeys(*bip_obj: Bip44Base*)

Bases: *HdWalletKeysBase*

HD wallet BIP keys class. It creates keys from a Bip object and stores them. Keys can be got individually, as dictionary or in JSON format.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

9.1.3 cardano

9.1.3.1 shelley

9.1.3.1.1 hd_wallet_cardano_shelley

Module for generating wallets based on Cardano Shelley.

class HdWalletCardanoShelley(*wallet_name: str*, *bip_obj: Bip44Base*, *mnemonic: str = ''*, *passphrase: str = ''*, *seed_bytes: bytes = b''*)

Bases: *HdWalletBase*

HD wallet Cardano Shelley class. It allows to generate a complete wallet based on Cardano Shelley specifications.

m_bip_obj: Bip44Base

Generate(***kwargs: Any*) → None

Generate wallet keys and addresses.

Parameters

- **acc_idx** (*int, optional*) – Account index (default: 0)
- **change_idx** (*HdWalletCardanoShelleyChanges, optional*) – Change index (default: external)
- **addr_num** (*int, optional*) – Number of addresses to be generated (default: 20)
- **addr_off** (*int, optional*) – Starting address index (default: 0)

IsWatchOnly() → bool

Get if the wallet is watch-only.

Returns :
bool: True if watch-only, false otherwise

9.1.3.1.2 hd_wallet_cardano_shelley_addr

Module with helper class for storing Cardano Shelley addresses.

class HdWalletCardanoShelleyAddresses(*shelley_obj: CardanoShelley*, *addr_num: int*, *addr_off: int*)

Bases: *HdWalletAddrBase*

HD wallet Cardano Shelley addresses class. It creates addresses from a CardanoShelley object and stores them. Addresses can be got individually, as dictionary or in JSON format.

m_addr_off: int

m_addr: List[Any]

m_dict_key_str_format: str

9.1.3.1.3 hd_wallet_cardano_shelley_enum

Module with enums for Cardano Shelley wallets.

class HdWalletCardanoShelleyDataTypes(*value*)

Bases: *HdWalletDataTypes*

Enumerative for wallet Cardano Shelley data types.

WALLET_NAME = 1

COIN_NAME = 2

MNEMONIC = 3

PASSPHRASE = 4

SEED_BYTES = 5

ACCOUNT_IDX = 6

CHANGE_IDX = 7

MASTER_KEY = 8

ACCOUNT_KEY = 9

STAKING_KEY = 10

ADDRESS_OFF = 11

ADDRESS = 12

class HdWalletCardanoShelleyKeyTypes(*value*)

Bases: *HdWalletKeyTypes*

Enumerative for wallet Cardano Shelley key types.

RAW_PRIV = 1

RAW_PUB = 2

ADDRESS = 3

9.1.3.1.4 hd_wallet_cardano_shelley_factory

Module for creating Cardano Shelley wallet factories.

class HdWalletCardanoShelleyFactory(*coin_type: Cip1852Coins*)

Bases: `object`

HD wallet Cardano Shelley factory class. It allows a HdWalletCardanoShelley to be created in different ways.

m_coin: Cip1852Coins

CreateRandom(*wallet_name: str*, *words_num: Optional[Bip39WordsNum] = None*, *lang: Bip39Languages = Bip39Languages.ENGLISH*) → *HdWalletBase*

Create wallet randomly.

Parameters

- **wallet_name** (*str*) – Wallet name
- **words_num** (*HdWalletCardanoShelleyWordsNum, optional*) – Words number (default: 15 for Icaurs, 24 for Ledger)
- **lang** (*HdWalletCardanoShelleyLanguages, optional*) – Language (default: English)

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
TypeError – If words number is not a HdWalletBipWordsNum enum or language is not a HdWalletBipLanguages enum

CreateFromMnemonic(*wallet_name: str*, *mnemonic: str*, *passphrase: str = ''*) → *HdWalletBase*

Create wallet from mnemonic.

Parameters

- **wallet_name** (*str*) – Wallet name
- **mnemonic** (*str*) – Mnemonic
- **passphrase** (*str, optional*) – Passphrase for protecting mnemonic, empty if not specified

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the mnemonic is not valid

CreateFromSeed(*wallet_name: str*, *seed_bytes: bytes*) → *HdWalletBase*

Create wallet from seed.

Parameters

- **wallet_name** (*str*) – Wallet name
- **seed_bytes** (*bytes*) – Seed bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the seed is not valid

CreateFromPrivateKey(*wallet_name: str*, *priv_key_bytes: bytes*) → *HdWalletBase*

Create wallet from private key. The key will be considered a master key.

Parameters

- **wallet_name** (*str*) – Wallet name
- **priv_key_bytes** (*bytes*) – Private key bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the private key is not valid

CreateFromPublicKey(*wallet_name: str*, *pub_key_bytes: bytes*) → *HdWalletBase*

Create wallet from public key. The key will be considered an account key in order to derive child keys.

Parameters

- **wallet_name** (*str*) – Wallet name
- **pub_key_bytes** (*bytes*) – Public key bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the public key is not valid

9.1.3.1.5 hd_wallet_cardano_shelley_keys

Module with helper class for storing Cardano Shelley keys.

class HdWalletCardanoShelleyKeysBase(*key_enum: Type[Enum]*)

Bases: *HdWalletKeysBase*

HD wallet Cardano Shelley keys base class. It contains some helper methods for Cardano Shelley keys classes.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

class HdWalletCardanoShelleyMasterKeys(*bip_obj: Bip44Base*)

Bases: *HdWalletCardanoShelleyKeysBase*

HD wallet Cardano Shelley master keys class. It creates keys from a CardanoShelley object and stores them. Keys can be got individually, as dictionary or in JSON format.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

class HdWalletCardanoShelleyStakingKeys(*shelley_obj: CardanoShelley*)

Bases: *HdWalletCardanoShelleyKeysBase*

HD wallet Cardano Shelley staking keys class. It creates keys from a CardanoShelley object and stores them. Keys can be got individually, as dictionary or in JSON format.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

class HdWalletCardanoShelleyDerivedKeys(*shelley_obj: CardanoShelley*)

Bases: *HdWalletCardanoShelleyKeysBase*

HD wallet Cardano Shelley derived keys class. It creates keys from a CardanoShelley object and stores them. Keys can be got individually, as dictionary or in JSON format.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

9.1.4 common

9.1.4.1 hd_wallet_addr_base

Module with base class for wallet addresses.

class HdWalletAddrBaseConst

Bases: `object`

Class container for HD wallet addresses base constants.

DICT_KEY_DEF_FORMAT: str = 'address_{:d}'

class HdWalletAddrBase(*addr_off: int*, *dict_key_str_format: Optional[str] = None*)

Bases: `ABC`

HD wallet addresses base class. It shall be inherited by wallet addresses classes.

m_addr_off: int

m_addr: List[Any]

m_dict_key_str_format: str

ToDict() → Dict[str, Any]

Get addresses as a dictionary.

Returns
Addresses as a dictionary

Return type
dict

ToJson(*json_indent: int = 4*) → str

Get addresses as string in JSON format.

Parameters
json_indent (*int, optional*) – Indent for JSON format, 4 by default

Returns
Addresses as string in JSON format

Return type
str

Count() → int

Get the addresses count.

Returns
Number of addresses

Return type
int

__getitem__(*addr_idx: int*) → Any

Get the specified address index.

Parameters
addr_idx (*int*) – Address index

Returns
Address

Return type
Any

__iter__() → Iterator[str]

Get the iterator to the current element.

Returns
Iterator to the current element

Return type
Iterator object

9.1.4.2 hd_wallet_base

Module with base class for wallet generators.

class HdWalletBase(*key_enum: Type[Enum]*)

Bases: *HdWalletEnumDict*, ABC

HD wallet base class. It shall be inherited by wallet classes.

abstract Generate(***kwargs: Any*) → None

Generate wallet keys and addresses.

Parameters
****kwargs** – Arbitrary arguments depending on the wallet type

abstract IsWatchOnly() → bool

Get if the wallet is watch-only.

Returns :
bool: True if watch-only, false otherwise

ToDict() → Dict[str, Any]

Get wallet data as a dictionary.

Returns
Wallet data as a dictionary

Return type
dict

`HasData`(*key:* HdWalletDataTypes) → bool

Get if the specified key is present.

Parameters
key (HdWalletKeyTypes) – Key

Returns
True if present, false otherwise

Return type
bool

Raises
TypeError – If the enumerative is not of the correct type

`GetData`(*key:* HdWalletDataTypes) → Optional[Any]

Get the specified key value.

Parameters
key (HdWalletKeyTypes) – Key

Returns
Key value None: If the key type is not found

Return type
str

Raises
TypeError – If the enumerative is not of the correct type

`m_key_enum: Type[Enum]`

`m_dict_data: Dict[str, Any]`

9.1.4.3 hd_wallet_data_types

Module for generic wallet data types enum.

`class HdWalletDataTypes`(*value*)

Bases: Enum

Base enum for wallet data types.

`class HdWalletKeyTypes`(*value*)

Bases: Enum

Base enum for wallet key types.

9.1.4.4 hd_wallet_enum_dict

Module with enum dictionary class.

class HdWalletEnumDict(*key_enum: Type[Enum]*)

Bases: `object`

HD wallet enum dictionary class. It allows a dictionary to be accessed using enum as string keys. It shall be inherited by all classes needed to use dictionaries to be accessed by enums.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

KeyEnum() → Type[Enum]

Get key enumerative type.

Returns
Key enumerative type

Return type
Enum

ToDict() → Dict[str, Any]

Get keys as a dictionary.

Returns
Keys as a dictionary

Return type
dict

ToJson(*json_indent: int = 4*) → str

Get keys as string in JSON format.

Parameters
json_indent (*int, optional*) – Indent for JSON format, 4 by default

Returns
Keys as string in JSON format

Return type
str

9.1.4.5 hd_wallet_keys_base

Module with base class for wallet keys.

class HdWalletKeysBase(*key_enum: Type[Enum]*)

Bases: *HdWalletEnumDict*

HD wallet keys base class. It shall be inherited by wallet keys classes.

HasKey(*key:* HdWalletKeyTypes) → bool

Get if the specified key is present.

Parameters
key (HdWalletKeyTypes) – Key

Returns
True if present, false otherwise

Return type
bool

Raises
TypeError – If the enumerative is not of the correct type

GetKey(*key:* HdWalletKeyTypes) → Optional[str]

Get the specified key value.

Parameters
key (HdWalletKeyTypes) – Key

Returns
Key value None: If the key type is not found

Return type
str

Raises
TypeError – If the enumerative is not of the correct type

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

9.1.5 electrum

9.1.5.1 v1

9.1.5.1.1 hd_wallet_electrum_v1

Module for generating wallets based on Electrum V1.

class HdWalletElectrumV1(*wallet_name: str*, *electrum_obj: ElectrumV1*, *mnemonic: str = ''*, *seed_bytes: bytes = b''*)

Bases: *HdWalletBase*

HD wallet Electrum V1 class. It allows to generate a wallet like Electrum V1.

m_electrum_obj: ElectrumV1

Generate(***kwargs: Any*) → None

Generate wallet keys and addresses.

Parameters

- **change_idx** (*int, optional*) – Change index (default: 0)
- **addr_num** (*int, optional*) – Number of addresses to be generated (default: 20)
- **addr_off** (*int, optional*) – Starting address index (default: 0)

IsWatchOnly() → bool

Get if the wallet is watch-only.

Returns :
bool: True if watch-only, false otherwise

9.1.5.1.2 hd_wallet_electrum_v1_addr

Module with helper class for storing Electrum V1 addresses.

class HdWalletElectrumV1Addresses(*electrum_obj: ElectrumV1*, *change_idx: int*, *addr_num: int*, *addr_off: int*)

Bases: *HdWalletAddrBase*

HD wallet Electrum V1 addresses class. It creates addresses from an ElectrumV1 object and stores them. Addresses can be got individually, as dictionary or in JSON format.

m_addr_off: int

m_addr: List[Any]

m_dict_key_str_format: str

9.1.5.1.3 hd_wallet_electrum_v1_enum

Module with enums for Electrum V1 wallets.

class HdWalletElectrumV1DataTypes(*value*)

Bases: *HdWalletDataTypes*

Enumerative for wallet Electrum V1 data types.

WALLET_NAME = 1

COIN_NAME = 2

MNEMONIC = 3

SEED_BYTES = 4

CHANGE_IDX = 5

MASTER_KEY = 6

ADDRESS_OFF = 7

ADDRESS = 8

class HdWalletElectrumV1KeyTypes(*value*)

Bases: *HdWalletKeyTypes*

Enumerative for wallet Electrum V1 key types.

RAW_PRIV = 1

WIF_PRIV = 2

RAW_PUB = 3

ADDRESS = 4

9.1.5.1.4 hd_wallet_electrum_v1_factory

Module for creating Electrum V1 wallet factories.

class HdWalletElectrumV1Factory

Bases: object

HD wallet Electrum V1 factory class. It allows a HdWalletElectrumV1 to be created in different ways.

CreateRandom(*wallet_name: str*, *words_num: ElectrumV1WordsNum = ElectrumV1WordsNum.WORDS_NUM_12*, *lang: ElectrumV1Languages = ElectrumV1Languages.ENGLISH*) → *HdWalletBase*

Create wallet randomly.

Parameters

- **wallet_name** (*str*) – Wallet name
- **words_num** (*HdWalletElectrumV1WordsNum, optional*) – Words number (default: 12)
- **lang** (*HdWalletElectrumV1Languages, optional*) – Language (default: English)

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
TypeError – If words number is not a HdWalletElectrumV1WordsNum enum or language is not a HdWalletElectrumV1Languages enum

CreateFromMnemonic(*wallet_name: str*, *mnemonic: str*) → *HdWalletBase*

Create wallet from mnemonic.

Parameters

- **wallet_name** (*str*) – Wallet name
- **mnemonic** (*str*) – Mnemonic

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the mnemonic is not valid

CreateFromSeed(*wallet_name: str*, *seed_bytes: bytes*) → *HdWalletBase*

Create wallet from seed.

Parameters

- **wallet_name** (*str*) – Wallet name
- **seed_bytes** (*bytes*) – Seed bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the seed is not valid

CreateFromPrivateKey(*wallet_name: str*, *priv_key_bytes: bytes*) → *HdWalletBase*

Create wallet from private key.

Parameters

- **wallet_name** (*str*) – Wallet name
- **priv_key_bytes** (*bytes*) – Private key bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the private key is not valid

CreateFromPublicKey(*wallet_name: str*, *pub_key_bytes: bytes*) → *HdWalletBase*

Create wallet from public key.

Parameters

- **wallet_name** (*str*) – Wallet name
- **pub_key_bytes** (*bytes*) – Public key bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the private key is not valid

9.1.5.1.5 hd_wallet_electrum_v1_keys

Module with helper class for storing Electrum V1 keys.

class HdWalletElectrumV1KeyUtils

Bases: object

Class container for HD wallet Electrum keys utility functions.

static PrivToWif(*priv_key: IPrivateKey*) → str

Encode private key to WIF.

Parameters
priv_key (*IPrivateKey object*) – Private key

Returns
WIF encoded key

Return type
str

class HdWalletElectrumV1MasterKeys(*electrum_obj: ElectrumV1*)

Bases: *HdWalletKeysBase*

HD wallet Electrum master keys class. It creates master keys from an ElectrumV1 object and stores them. Keys can be got individually, as dictionary or in JSON format.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

class HdWalletElectrumV1DerivedKeys(*electrum_obj: ElectrumV1*, *change_idx: int*, *addr_num: int*, *addr_off: int*)

Bases: *HdWalletKeysBase*

HD wallet Electrum derived keys class. It creates derived from an Electrum object and stores them. Keys can be got individually, as dictionary or in JSON format.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

9.1.5.2 v2

9.1.5.2.1 hd_wallet_electrum_v2

Module for generating wallets based on Electrum V2.

class HdWalletElectrumV2(*wallet_name: str*, *electrum_obj: ElectrumV2Base*, *mnemonic: str = ''*, *passphrase: str = ''*, *seed_bytes: bytes = b''*)

Bases: *HdWalletBase*

HD wallet Electrum V2 class. It allows to generate a wallet like Electrum V2.

m_electrum_obj: ElectrumV2Base

Generate(***kwargs: Any*) → None

Generate wallet keys and addresses.

Parameters

- **change_idx** (*int, optional*) – Change index (default: 0)
- **addr_num** (*int, optional*) – Number of addresses to be generated (default: 20)
- **addr_off** (*int, optional*) – Starting address index (default: 0)

IsWatchOnly() → bool

Get if the wallet is watch-only.

Returns :

bool: True if watch-only, false otherwise

9.1.5.2.2 hd_wallet_electrum_v2_addr

Module with helper class for storing Electrum V2 addresses.

class HdWalletElectrumV2Addresses(*electrum_obj: ElectrumV2Base*, *change_idx: int*, *addr_num: int*, *addr_off: int*)

Bases: *HdWalletAddrBase*

HD wallet Electrum V2 addresses class. It creates addresses from an ElectrumV2 object and stores them. Addresses can be got individually, as dictionary or in JSON format.

m_addr_off: int

m_addr: List[Any]

m_dict_key_str_format: str

9.1.5.2.3 hd_wallet_electrum_v2_enum

Module with enums for Electrum V2 wallets.

class HdWalletElectrumV2DataTypes(*value*)

Bases: *HdWalletDataTypes*

Enumerative for wallet Electrum V2 data types.

WALLET_NAME = 1

COIN_NAME = 2

MNEMONIC = 3

PASSPHRASE = 4

SEED_BYTES = 5

CHANGE_IDX = 6

MASTER_KEY = 7

ADDRESS_OFF = 8

ADDRESS = 9

class HdWalletElectrumV2KeyTypes(*value*)

Bases: *HdWalletKeyTypes*

Enumerative for wallet Electrum V2 key types.

EX_PRIV = 1

RAW_PRIV = 2

WIF_PRIV = 3

EX_PUB = 4

RAW_PUB = 5

ADDRESS = 6

9.1.5.2.4 hd_wallet_electrum_v2_factory

Module for creating Electrum V2 wallet factories.

class HdWalletElectrumV2Factory(*mnemonic_type: ElectrumV2MnemonicTypes*)

Bases: `object`

HD wallet Electrum V2 factory class. It allows a HdWalletElectrumV2 to be created in different ways.

m_mnemonic_type: ElectrumV2MnemonicTypes

m_electrum_cls: Type[ElectrumV2Base]

CreateRandom(*wallet_name: str*, *words_num: ElectrumV2WordsNum = ElectrumV2WordsNum.WORDS_NUM_12*, *lang: ElectrumV2Languages = ElectrumV2Languages.ENGLISH*) → *HdWalletBase*

Create wallet randomly.

Parameters

- **wallet_name** (*str*) – Wallet name
- **words_num** (*HdWalletElectrumV2WordsNum, optional*) – Words number (default: 12)
- **lang** (*HdWalletElectrumV2Languages, optional*) – Language (default: English)

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
TypeError – If words number is not a HdWalletElectrumV2WordsNum enum or language is not a HdWalletElectrumV2Languages enum

CreateFromMnemonic(*wallet_name: str*, *mnemonic: str*, *passphrase: str = ''*) → *HdWalletBase*

Create wallet from mnemonic.

Parameters

- **wallet_name** (*str*) – Wallet name
- **mnemonic** (*str*) – Mnemonic
- **passphrase** (*str, optional*) – Passphrase for protecting mnemonic, empty if not specified

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the mnemonic is not valid

CreateFromSeed(*wallet_name: str*, *seed_bytes: bytes*) → *HdWalletBase*

Create wallet from seed.

Parameters

- **wallet_name** (*str*) – Wallet name

- **seed_bytes** (*bytes*) – Seed bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the seed is not valid

CreateFromExtendedKey(*wallet_name: str*, *ex_key_str: str*) → *HdWalletBase*

Create wallet from extended key.

Parameters

- **wallet_name** (*str*) – Wallet name
- **ex_key_str** (*str*) – Extended key string

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the extended key is public or not valid

9.1.5.2.5 hd_wallet_electrum_v2_keys

Module with helper class for storing Electrum V2 keys.

class HdWalletElectrumV1KeyUtils

Bases: object

Class container for HD wallet Electrum keys utility functions.

static PrivToWif(*priv_key: Bip32PrivateKey*) → str

Encode private key to WIF.

Parameters
priv_key (*Bip32PrivateKey object*) – BIP32 p rivate key

Returns
WIF encoded key

Return type
str

class HdWalletElectrumV2MasterKeys(*electrum_obj: ElectrumV2Base*)

Bases: *HdWalletKeysBase*

HD wallet Electrum master keys class. It creates master keys from an ElectrumV2 object and store them. Keys can be got individually, as dictionary or in JSON format.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

class HdWalletElectrumV2DerivedKeys(*electrum_obj: ElectrumV2Base*, *change_idx: int*, *addr_num: int*, *addr_off: int*)

Bases: *HdWalletKeysBase*

HD wallet Electrum derived keys class. It creates derived keys from an Electrum object and store them. Keys can be got individually, as dictionary or in JSON format.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

9.1.6 monero

9.1.6.1 hd_wallet_monero

Module for generating Monero wallets.

class HdWalletMonero(*wallet_name: str*, *monero_obj: Monero*, *mnemonic: str = ''*, *seed_bytes: bytes = b''*)

Bases: *HdWalletBase*

HD wallet Monero class. It allows to generate a Monero wallet like the official one.

m_monero_obj: Monero

Generate(***kwargs: Any*) → None

Generate wallet keys and addresses.

Parameters

- **acc_idx** (*int, optional*) – Account index (default: 0)
- **subaddr_num** (*int, optional*) – Subaddress number (default: 0)
- **subaddr_off** (*int, optional*) – Starting subaddress index (default: 0)

IsWatchOnly() → bool

Get if the wallet is watch-only.

Returns :
bool: True if watch-only, false otherwise

9.1.6.2 hd_wallet_monero_enum

Module with enums for Monero wallets.

class HdWalletMoneroDataTypes(*value*)

Bases: *HdWalletDataTypes*

Enumerative for wallet Monero data types.

WALLET_NAME = 1

COIN_NAME = 2

MNEMONIC = 3

SEED_BYTES = 4

KEY = 5

ACCOUNT_IDX = 6

SUBADDRESS_OFF = 7

SUBADDRESS = 8

class HdWalletMoneroKeyTypes(*value*)

Bases: *HdWalletKeyTypes*

Enumerative for wallet Monero key types.

PRIV_SPEND = 1

PRIV_VIEW = 2

PUB_SPEND = 3

PUB_VIEW = 4

PRIMARY_ADDRESS = 5

9.1.6.3 hd_wallet_monero_factory

Module for creating Monero wallet factories.

class HdWalletMoneroFactory(*coin_type: MoneroCoins = MoneroCoins.MONERO_MAINNET*)

Bases: `object`

HD wallet Monero factory class. It allows a HdWalletMonero to be created in different ways.

m_monero_coin: MoneroCoins

CreateRandom(*wallet_name: str*, *words_num: MoneroWordsNum = MoneroWordsNum.WORDS_NUM_25*, *lang: MoneroLanguages = MoneroLanguages.ENGLISH*) → *HdWalletBase*

Create wallet randomly.

Parameters

- **wallet_name** (*str*) – Wallet name
- **words_num** (*HdWalletMoneroWordsNum, optional*) – Words number (default: 25)
- **lang** (*HdWalletMoneroLanguages, optional*) – Language (default: English)

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
TypeError – If words number is not a HdWalletMoneroWordsNum enum or language is not a HdWalletMoneroLanguages enum

CreateFromMnemonic(*wallet_name: str*, *mnemonic: str*) → *HdWalletBase*

Create wallet from mnemonic.

Parameters

- **wallet_name** (*str*) – Wallet name

- **mnemonic** (*str*) – Mnemonic

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the mnemonic is not valid

CreateFromSeed(*wallet_name: str*, *seed_bytes: bytes*) → *HdWalletBase*

Create wallet from seed.

Parameters

- **wallet_name** (*str*) – Wallet name
- **seed_bytes** (*bytes*) – Seed bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the seed is not valid

CreateFromPrivateKey(*wallet_name: str*, *priv_skey_bytes: bytes*) → *HdWalletBase*

Create wallet from private spend key.

Parameters

- **wallet_name** (*str*) – Wallet name
- **priv_skey_bytes** (*bytes*) – Private spend key bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the private key is not valid

CreateFromWatchOnly(*wallet_name: str*, *priv_vkey_bytes: bytes*, *pub_skey_bytes: bytes*) → *HdWalletBase*

Create wallet from private view key and public spend key (i.e. watch-only wallet).

Parameters

- **wallet_name** (*str*) – Wallet name
- **priv_vkey_bytes** (*bytes*) – Private view key bytes
- **pub_skey_bytes** (*bytes*) – Public spend key bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the public key is not valid

9.1.6.4 hd_wallet_monero_keys

Module with helper class for storing Monero keys.

class HdWalletMoneroKeys(*monero_obj: Monero*)

Bases: *HdWalletKeysBase*

HD wallet Monero keys class. It creates keys from a Monero object and stores them. Keys can be got individually, as dictionary or in JSON format.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

9.1.6.5 hd_wallet_monero_subaddr

Module with helper class for storing Monero subaddresses.

class HdWalletMoneroSubaddressesConst

Bases: object

Class container for HD wallet Monero subaddresses constants.

DICT_KEY_FORMAT: str = 'subaddress_{:d}'

class HdWalletMoneroSubaddresses(*monero_obj: Monero*, *acc_idx: int*, *subaddr_num: int*, *subaddr_off: int*)

Bases: *HdWalletAddrBase*

HD wallet Monero subaddresses class. It creates subaddresses from a Monero object and stores them. Subaddresses can be got individually, as dictionary or in JSON format.

m_addr_off: int

m_addr: List[Any]

m_dict_key_str_format: str

9.1.7 saver

9.1.7.1 hd_wallet_saver

Module for saving wallets to file.

class HdWalletSaver(*hd_wallet:* HdWalletBase)

Bases: object

HD wallet saver class. It saves a wallet to file.

m_hd_wallet: *HdWalletBase*

SaveToFile(*file_path: str*) → None

Save wallet to file in JSON format.

Parameters

file_path (*str*) – File path

9.1.8 substrate

9.1.8.1 hd_wallet_substrate

Module for generating Substrate wallets.

class HdWalletSubstrate(*wallet_name: str*, *substrate_obj: Substrate*, *mnemonic: str = ''*, *passphrase: str = ''*, *seed_bytes: bytes = b''*)

Bases: *HdWalletBase*

HD wallet Substrate class. It allows to generate a Substrate wallet like the official one.

m_substrate_obj: Substrate

Generate(***kwargs: Any*) → None

Generate wallet keys and addresses.

Parameters
path (*str, optional*) – Derivation path (default: empty)

IsWatchOnly() → bool

Get if the wallet is watch-only.

Returns :
bool: True if watch-only, false otherwise

9.1.8.2 hd_wallet_substrate_enum

Module with enums for Substrate wallets.

class HdWalletSubstrateDataTypes(*value*)

Bases: *HdWalletDataTypes*

Enumerative for wallet Substrate data types.

WALLET_NAME = 1

COIN_NAME = 2

MNEMONIC = 3

PASSPHRASE = 4

SEED_BYTES = 5

PATH = 6

KEY = 7

class HdWalletSubstrateKeyTypes(*value*)

Bases: *HdWalletKeyTypes*

Enumerative for wallet Substrate key types.

PRIV = 1

PUB = 2

ADDRESS = 3

9.1.8.3 hd_wallet_substrate_factory

Module for creating Substrate wallet factories.

class HdWalletSubstrateFactory(*coin_type: SubstrateCoins*)

Bases: `object`

HD wallet Substrate factory class. It allows a HdWalletSubstrate to be created in different ways.

m_substrate_coin: SubstrateCoins

CreateRandom(*wallet_name: str*, *words_num: Bip39WordsNum = Bip39WordsNum.WORDS_NUM_24*, *lang: Bip39Languages = Bip39Languages.ENGLISH*) → *HdWalletBase*

Create wallet randomly.

Parameters

- **wallet_name** (*str*) – Wallet name
- **words_num** (*HdWalletSubstrateWordsNum, optional*) – Words number (default: 24)
- **lang** (*HdWalletSubstrateLanguages, optional*) – Language (default: English)

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
TypeError – If words number is not a HdWalletSubstrateWordsNum enum or language is not a HdWalletSubstrateLanguages enum

CreateFromMnemonic(*wallet_name: str*, *mnemonic: str*, *passphrase: str = ''*) → *HdWalletBase*

Create wallet from mnemonic.

Parameters

- **wallet_name** (*str*) – Wallet name
- **mnemonic** (*str*) – Mnemonic
- **passphrase** (*str, optional*) – Passphrase for protecting mnemonic, empty if not specified

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the mnemonic is not valid

CreateFromSeed(*wallet_name: str*, *seed_bytes: bytes*) → *HdWalletBase*

Create wallet from seed.

Parameters

- **wallet_name** (*str*) – Wallet name
- **seed_bytes** (*bytes*) – Seed bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the seed is not valid

CreateFromPrivateKey(*wallet_name: str*, *priv_key_bytes: bytes*) → *HdWalletBase*

Create wallet from private key.

Parameters

- **wallet_name** (*str*) – Wallet name
- **priv_key_bytes** (*bytes*) – Private key bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the private key is not valid

CreateFromPublicKey(*wallet_name: str*, *pub_key_bytes: bytes*) → *HdWalletBase*

Create wallet from public key.

Parameters

- **wallet_name** (*str*) – Wallet name
- **pub_key_bytes** (*bytes*) – Public key bytes

Returns
HdWalletBase object

Return type
HdWalletBase object

Raises
ValueError – If the public key is not valid

9.1.8.4 hd_wallet_substrate_keys

Module with helper class for storing Substrate keys.

class HdWalletSubstrateKeys(*substrate_obj: Substrate*)

Bases: *HdWalletKeysBase*

HD wallet Substrate keys class. It creates keys from a Substrate object and stores them. Keys can be got individually, as dictionary or in JSON format.

m_key_enum: Type[Enum]

m_dict_data: Dict[str, Any]

9.1.9 utils

9.1.9.1 utils

Module with some utility functions.

class Utils

Bases: object

Class container for utility functions.

static BytesToHexString(*data_bytes: bytes*, *encoding: str = 'utf-8'*) → str

Convert bytes to hex string.

Parameters

- **data_bytes** (*bytes*) – Data bytes
- **encoding** (*str, optional*) – Encoding type

Returns
Bytes converted to hex string

Return type
str

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

G

H

I

K

M

P

R

S

T

U

W