pons

Release 0.8.0-dev

Bogdan Opanchuk

Mar 16, 2024

CONTENTS:

A quick usage example:

```
import trio

from eth_account import Account
from pons import Client, HTTPProvider, AccountSigner, Address, Amount

async def main():

    provider = HTTPProvider("<your provider's https endpoint>")
    client = Client(provider)

    acc = Account.from_key("0x<your secret key>")
    signer = AccountSigner(acc)

    async with client.session() as session:
        my_balance = await session.eth_get_balance(signer.address)
        print("My balance:", my_balance.as_ether(), "ETH")

        another_address = Address.from_hex("0x<another_address>")
        await session.transfer(signer, another_address, Amount.ether(1.5))

        another_balance = await session.eth_get_balance(another_address)
        print("Another balance:", another_balance.as_ether(), "ETH")

trio.run(main)
```

```
My balance: 100.0 ETH
Another balance: 1.5 ETH
```

For more usage information, proceed to *Tutorial*.

CHAPTER

ONE

TUTORIAL

1.1 Async support

While the examples and tests use `trio`, `pons` is `anyio`-based and supports all the corresponding backends.

1.2 Sessions

All calls to the provider in `pons` happen within a session. It translates to the usage of a single session in the backend HTTP request library, so the details are implementation-dependent, but in general it means that multiple requests will happen faster. For example, in a session an SSL handshake only happens once, and it is a somewhat slow process.

Correspondingly, all the main functionality of the library is concentrated in the `ClientSession` class.

1.3 Signers

Any operation that entails writing information into the blockchain takes a `Signer` object. For now, only signers created from `eth_account.Account` are supported, but one can define their own class backed by, say, a hardware signer, using the abstract `Signer` class.

1.4 Amounts and addresses

Native currency amounts and network addresses are typed in `pons`. All methods expect and return only `Amount` and `Address` objects — no integers or strings allowed.

In an application using `pons` one can superclass these classes to distinguish between different types of currencies, or addresses from different networks. Note though that all the arithmetic and comparison functions require **strict** type equality and raise an exception if it is not the case, to protect from accidental usage of addresses/amounts from wrong domains.

1.5 Contract ABI

Contract ABI can be declared in two different ways in pons. The first one can be used when you have a JSON ABI definition, for example installed as a JS package, or obtained from compiling a contract.

```
from pons import ContractABI

cabi = ContractABI.from_json(json_abi)
print(cabi)
```

This will show a brief summary of the ABI in a C-like code.

```
{
    constructor(uint256 _v1, uint256 _v2) nonpayable
    fallback() nonpayable
    receive() payable
    function getState(uint256 _x) view returns (uint256)
    function testStructs((uint256,uint256) inner_in, ((uint256,uint256),uint256) outer_
↪in) view returns ((uint256,uint256) inner_out, ((uint256,uint256),uint256) outer_out)
    function v1() view returns (uint256)
    function v2() view returns (uint256)
    function setState(uint256 _v1) nonpayable
}
```

Alternatively, one can define only the methods they need directly in Python code:

```
from pons import ContractABI, abi, Constructor, Method, Mutability

inner_struct = abi.struct(inner1=abi.uint(256), inner2=abi.uint(256))
outer_struct = abi.struct(inner=inner_struct, outer1=abi.uint(256))
cabi = ContractABI(
    constructor=Constructor(inputs=dict(_v1=abi.uint(256), _v2=abi.uint(256))),
    methods=[
        Method(
            name='setState',
            mutability=Mutability.NONPAYABLE,
            inputs=dict(_v1=abi.uint(256)))
        Method(
            name='getState',
            mutability=Mutability.VIEW,
            inputs=dict(_x=abi.uint(256)),
            outputs=abi.uint(256)),
        Method(
            name='testStructs',
            mutability=Mutability.VIEW,
            inputs=dict(inner_in=inner_struct, outer_in=outer_struct),
            outputs=dict(inner_out=inner_struct, outer_out=outer_struct),
            )
        ]
    )

print(cabi)
```

```
{
    constructor(uint256 _v1, uint256 _v2) nonpayable
    function getState(uint256 _x) view returns (uint256)
    function testStructs((uint256,uint256) inner_in, ((uint256,uint256),uint256) outer_
↪in) view returns ((uint256,uint256) inner_out, ((uint256,uint256),uint256) outer_out)
    function setState(uint256 _v1) nonpayable
}
```

1.6 Contract methods

All the enumerated methods have corresponding objects that can be accessed via `ContractABI` fields (see the API reference for details). For example,

```
print(cabi.method.getState)
```

```
function getState(uint256 _x) view returns (uint256)
```

With a specific method object one can create a contract call by, naturally, calling the object. The arguments are processed the same as in Python functions, so one can either use positional arguments, keyword ones (if the parameter names are present in the contract ABI), or mix the two.

```
call = cabi.method.getState(1)
call = cabi.method.getState(_x=1)
```

Note that the arguments are checked and encoded on call creation, so any inconsistency will result in a raised exception:

```
call = cabi.method.getState(1, 2)
```

```
Traceback (most recent call last):
...
TypeError: too many positional arguments
```

```
call = cabi.method.getState("a")
```

```
Traceback (most recent call last):
...
TypeError: `uint256` must correspond to an integer, got str
```

1.7 Deploying contracts

In order to deploy a contract one needs its ABI and bytecode. At the moment `pons` does not expose the compiler interface, so it has to come from a third party library, for example py-solcx. With that, create a `CompiledContract` object and use `deploy()`:

```
compiled_contract = CompiledContract(cabi, bytecode)
deployed_contract = await session.deploy(signer, compiled_contract.constructor(arg1,␣
↪arg2))
```

This will result in a `DeployedContract` object encapsulating the contract address and its ABI and allowing one to interact with the contract.

Alternatively, a `DeployedContract` object can be created with a known address if the contract is already deployed:

```
deployed_contract = DeployedContract(cabi, Address.from_hex("0x<contract_address>"))
```

1.8 Interacting with deployed contracts

A `DeployedContract` object wraps all ABI method objects into "bound" state, similarly to how Python methods are bound to class instances. It means that all the method calls created from this object have the contract address inside them, so that it does not need to be provided every time.

For example, to call a non-mutating contract method via `eth_call()`:

```
call = deployed_contract.method.getState(1)
result = await session.eth_call(call)
```

Note that when the `ContractABI` object is created from the JSON ABI, even if the method returns a single value, it is still represented as a list of one element in the JSON, so the `result` will be a list too. If the ABI is declared programmatically, one can provide a single output value instead of the list, and then `pons` will unpack that list.

Naturally, a mutating call requires a signer to be provided:

```
call = deployed_contract.method.setState(1)
await session.transact(signer, call)
```

CHAPTER

TWO

API

2.1 Clients

2.2 Providers

2.3 Fallback providers

2.4 Errors

class pons._provider.**RPCError**(*code: ErrorCode*, *message: str*, *data: None | bytes = None*)

A wrapper for a call execution error returned as a proper RPC response.

class pons._client.**ContractPanicReason**(*value*)

Reasons leading to a contract call panicking.

ASSERTION = 1

If you call assert with an argument that evaluates to `false`.

COMPILER = 0

Used for generic compiler inserted panics.

DIVISION_BY_ZERO = 18

If you divide or modulo by zero (e.g. `5 / 0` or `23 % 0`).

EMPTY_ARRAY = 49

If you call `.pop()` on an empty array.

INVALID_ENCODING = 34

If you access a storage byte array that is incorrectly encoded.

INVALID_ENUM_VALUE = 33

If you convert a value that is too big or negative into an `enum` type.

OUT_OF_BOUNDS = 50

If you access an array, `bytesN` or an array slice at an out-of-bounds or negative index (i.e. `x[i]` where `i >= x.length` or `i < 0`).

OUT_OF_MEMORY = 65

If you allocate too much memory or create an array that is too large.

OVERFLOW = 17

If an arithmetic operation results in underflow or overflow outside of an `unchecked { ... }` block.

UNKNOWN = -1

Unknown panic code.

ZERO_DEREFERENCE = 81

If you call a zero-initialized variable of internal function type.

2.5 Signers

2.6 Contract ABI

2.7 Testing utilities

`pons` exposes several types useful for testing applications that connect to Ethereum RPC servers. Not intended for the production environment.

2.8 Secondary classes

The instances of these classes are not created by the user directly, but rather found as return values, or attributes of other objects.

2.9 Utility classes

class pons._contract_abi.**Methods**

Bases: `Generic`[`MethodType`].

A holder for named methods which can be accessed as attributes, or iterated over.

__getattr__(*method_name: str*) → *MethodType*

Returns the method by name.

__iter__() → Iterator[*MethodType*]

Returns the iterator over all methods.

class pons._contract_abi.**MethodType**

Generic method type parameter.

class pons._contract_abi.**Signature**

Generalized signature of either inputs or outputs of a method.

property canonical_form: str

Returns the signature serialized in the canonical form as a string.

class pons._contract_abi.**Method**(*name: str*, *mutability: Mutability*, *inputs: Mapping[str,* Type*]* | *Sequence[*Type*]*, *outputs: None* | *Mapping[str,* Type*]* | *Sequence[*Type*]* | Type = *None*)

A contract method.

Note: If the name of a parameter (input or output) given to the constructor matches a Python keyword, _ will be appended to it.

---

decode_output(*output_bytes: bytes*) → Any

Decodes the output from ABI-packed bytes.

classmethod from_json(*method_entry: dict[str, Any]*) → *Method*

Creates this object from a JSON ABI method entry.

property inputs: *Signature*

The input signature of this method.

mutating: bool

Whether this method may mutate the contract state.

property name: str

The name of this method.

outputs: *Signature*

Method's output signature.

payable: bool

Whether this method is marked as payable.

property selector: bytes

Method's selector.

2.10 Compiled and deployed contracts

2.11 Entities

class pons._entities.**CustomAmount**

A type derived from Amount.

class pons._entities.**CustomAddress**

A type derived from Address.

class pons._entities.**TxReceipt**

Transaction receipt.

block_hash: BlockHash

Hash of the block including this transaction.

block_number: int

Block number including this transaction.

contract_address: None | Address

If it was a successful deployment transaction, contains the address of the deployed contract.

cumulative_gas_used: int
The total amount of gas used when this transaction was executed in the block.

effective_gas_price: Amount
The actual value per gas deducted from the sender's account.

from_: Address
Address of the sender.

gas_used: int
The amount of gas used by the transaction.

logs: tuple[*LogEntry*, ...]
An array of log objects generated by this transaction.

status: int
1 if the transaction was successful, 0 otherwise.

property succeeded: bool
`True` if the transaction succeeded.

to: None | Address
Address of the receiver. `None` when the transaction is a contract creation transaction.

transaction_hash: TxHash
Hash of the transaction.

transaction_index: int
Integer of the transaction's index position in the block.

type_: int
Transaction type: 0 for legacy transactions, 2 for EIP1559 transactions.

class pons._entities.**BlockInfo**
Block info.

base_fee_per_gas: Amount
Base fee per gas in this block.

difficulty: int
Block's difficulty.

gas_limit: int
Block's gas limit.

gas_used: int
Gas used for the block.

hash_: None | BlockHash
Block hash. `None` for pending blocks.

miner: None | Address
Block's miner. `None` for pending blocks.

nonce: None | int
Block's nonce. `None` for pending blocks.

number: int
Block number.

parent_hash: BlockHash

Parent block's hash.

size: int

Block size.

timestamp: int

Block's timestamp.

total_difficulty: None | int

Block's totat difficulty. None for pending blocks.

transactions: tuple[*TxInfo*, ...] | tuple[TxHash, ...]

A list of transaction hashes in this block, or a list of details of transactions in this block, depending on what was requested.

class pons._entities.**TxInfo**

Transaction info.

block_hash: None | BlockHash

The hash of the block this transaction belongs to. None for pending transactions.

block_number: int

The number of the block this transaction belongs to. May be a pending block.

from_: Address

Transaction sender.

gas: int

Gas used by the transaction.

gas_price: Amount

Gas price used by the transaction.

hash_: TxHash

Transaction hash.

input_: None | bytes

The data sent along with the transaction.

max_fee_per_gas: None | Amount

maxFeePerGas value specified by the sender. Only for EIP1559 transactions.

max_priority_fee_per_gas: None | Amount

maxPriorityFeePerGas value specified by the sender. Only for EIP1559 transactions.

nonce: int

Transaction nonce.

to: None | Address

Transaction recipient. None when it's a contract creation transaction.

transaction_index: None | int

Transaction index. None for pending transactions.

type_: int

Transaction type: 0 for legacy transactions, 2 for EIP1559 transactions.

value: Amount

Associated funds.

class pons._entities.**BlockFilter**

BlockFilter(**id_**: pons._entities.BlockFilterId, provider_path: tuple[int, ...])

class pons._entities.**PendingTransactionFilter**

PendingTransactionFilter(**id_**: pons._entities.PendingTransactionFilterId, provider_path: tuple[int, ...])

class pons._entities.**LogFilter**

LogFilter(**id_**: pons._entities.LogFilterId, provider_path: tuple[int, ...])

class pons._entities.**LogTopic**

A log topic for log filtering.

class pons._entities.**LogEntry**

Log entry metadata.

address: Address

The contract address from which this log originated.

block_hash: BlockHash

Hash of the block where this log was in.

block_number: int

The block number where this log was.

data: bytes

ABI-packed non-indexed arguments of the event.

log_index: int

Log's position in the block.

removed: bool

`True` if log was removed, due to a chain reorganization. `False` if it is a valid log.

topics: tuple[*LogTopic*, ...]

Values of indexed event fields. For a named event, the first topic is the event's selector.

transaction_hash: TxHash

Hash of the transactions this log was created from.

transaction_index: int

Transaction's position in the block.

class JSON

A JSON-ifiable object (`bool`, `int`, `float`, `str`, `None`, iterable of `JSON`, or mapping of `str` to `JSON`).

2.12 Solidity types

Type aliases are exported from the `abi` submodule. Arrays can be obtained from `Type` objects by indexing them (either with an integer for a fixed-size array, or with `...` for a variable-sized array).

Helper aliases are exported from `pons.abi` submodule:

`pons.abi.`**`uint`**(*bits: int*) → *UInt*

Returns the `uint<bits>` type.

`pons.abi.`**`int`**(*bits: int*) → *Int*

Returns the `int<bits>` type.

`pons.abi.`**`bytes`**(*size: None | int = None*) → *Bytes*

Returns the `bytes<size>` type, or `bytes` if `size` is `None`.

`pons.abi.`**`address`**: *AddressType*

`address` type.

`pons.abi.`**`string`**: *String*

`string` type.

`pons.abi.`**`bool`**: *Bool*

`bool` type.

`pons.abi.`**`struct`**(***kwargs: Type*) → *Struct*

Returns the structure type with given fields.

Actual type objects, for reference:

`class` `pons._abi_types.`**`Type`**

The base type for Solidity types.

`class` `pons._abi_types.`**`UInt`**(*bits: int*)

Corresponds to the Solidity `uint<bits>` type.

`class` `pons._abi_types.`**`Int`**(*bits: int*)

Corresponds to the Solidity `int<bits>` type.

`class` `pons._abi_types.`**`Bytes`**(*size: None | int = None*)

Corresponds to the Solidity `bytes<size>` type.

`class` `pons._abi_types.`**`AddressType`**

Corresponds to the Solidity `address` type. Not to be confused with `Address` which represents an address value.

`class` `pons._abi_types.`**`String`**

Corresponds to the Solidity `string` type.

`class` `pons._abi_types.`**`Bool`**

Corresponds to the Solidity `bool` type.

`class` `pons._abi_types.`**`Struct`**(*fields: Mapping[str, Type]*)

Corresponds to the Solidity struct type.

CHAPTER

THREE

CHANGELOG

3.1 0.8.0 (unreleased)

3.1.1 Changed

- Added an explicit `typing_extensions` dependency. (PR_57)
- Various boolean arguments are now keyword-only to prevent usage errors. (PR_57)
- Field names clashing with Python built-ins (`hash`, `type`, `id`) are suffixed with an underscore. (PR_57)
- `AccountSigner` takes `LocalSigner` specifically and not just any `BaseSigner`. (PR_62)
- `ClientSession.estimate_transact()` and `estimate_deploy()` now require a `sender_address` parameter. (PR_62)
- Switched to `alysis` from `eth-tester` for the backend of `LocalProvider`. (PR_70)
- Bumped the minimum Python version to 3.10. (PR_72)
- The entities are now dataclasses instead of namedtuples. (PR_75)

3.1.2 Added

- `Client.transact()` takes an optional `return_events` argument, allowing one to get "return values" from the transaction via events. (PR_52)
- Exposed `ClientSession`, `ConstructorCall`, `MethodCall`, `EventFilter`, `BoundConstructor`, `BoundConstructorCall`, `BoundMethod`, `BoundMethodCall`, `BoundEvent`, `BoundEventFilter` from the top level. (PR_56)
- Various methods that had a default `Amount(0)` for a parameter can now take `None`. (PR_57)
- Support for overloaded methods via `MultiMethod`. (PR_59)
- Expose `HTTPProviderServer`, `LocalProvider`, `compile_contract_file` that can be used for tests of Ethereum-using applications. These are gated behind optional features. (PR_54)
- `LocalProvider.take_snapshot()` and `revert_to_snapshot()`. (PR_61)
- `AccountSigner.private_key` property. (PR_62)
- `LocalProvider.add_account()` method. (PR_62)
- An optional `sender_address` parameter of `ClientSession.eth_call()`. (PR_62)
- Expose `Provider` at the top level. (PR_63)

- `eth_getCode` support (as `ClientSession.eth_get_code()`). (PR_64)
- `eth_getStorageAt` support (as `ClientSession.eth_get_storage_at()`). (PR_64)
- Support for the `logs` field in `TxReceipt`. (PR_68)
- `ClientSession.eth_get_logs()` and `eth_get_filter_logs()`. (PR_68)
- Support for a custom block number in gas estimation methods. (PR_70)

3.1.3 Fixed

- Process unnamed arguments in JSON entries correctly (as positional arguments). (PR_51)
- More robust error handling in HTTP provider. (PR_63)
- The transaction tip being set larger than the max gas price (which some providers don't like). (PR_64)
- Decoding error when fetching pending transactions. (PR_65)
- Decoding error when fetching pending blocks. (PR_67)
- Get the default nonce based on the pending block, not the latest one. (PR_68)
- Using `eth_getLogs` instead of creating a filter in `transact()`. (PR_70)
- Expect the block number to be non-null even for pending blocks, since that's what providers return. (PR_70)

3.2 0.7.0 (09-07-2023)

3.2.1 Changed

- `ReadMethod` and `WriteMethod` were merged into `Method` (with the corresponding merge of `ContractABI` routing objects and various bound calls). (PR_50)

3.2.2 Added

- `Block.SAFE` and `Block.FINALIZED` values. (PR_48)
- `FallbackProvider`, two strategies for it (`CycleFallback` and `PriorityFallback`), and a framework for creating user-defined strategies (`FallbackStrategy` and `FallbackStrategyFactory`). (PR_49)
- `Mutability` enum for defining contract method mutability. (PR_50)

3.3 0.6.0 (11-05-2023)

3.3.1 Changed

- Parameter names and fields coinciding with Python keywords have _ appended to them on the creation of ABI objects. (PR_47)

3.3.2 Added

- Added support for Python 3.11. (PR_47)

3.3.3 Fixed

- Support the existence of outputs in the JSON ABI of a mutating method. (PR_47)

3.4 0.5.1 (14-11-2022)

3.4.1 Fixed

- A bug in processing keyword arguments to contract calls. (PR_42)

3.5 0.5.0 (14-09-2022)

3.5.1 Changed

- Bumped dependencies: `eth-account>=0.6`, `eth-utils>=2`, `eth-abi>=3`. (PR_40)

3.5.2 Fixed

- Return type of classmethods of `Amount` and `Address` now provides correct information to `mypy` in dependent projects. (PR_37)

3.6 0.4.2 (05-06-2022)

3.6.1 Added

- `__repr__`/`__eq__`/`__hash__` implementations for multiple entities. (PR_32)
- `eth_get_transaction_by_hash()`, `eth_block_number()`, `eth_get_block_by_hash()`, `eth_get_block_by_number()` and corresponding entities. (PR_32)
- `eth_new_block_filter()`, `eth_new_pending_transaction_filter()`, `eth_new_filter()`, `eth_get_filter_changes()` for low-level event filtering support. (PR_32)
- `iter_blocks()`, `iter_pending_transactions()`, `iter_events()` for high-level event filtering support. (PR_32)
- More fields in `TxReceipt`. (PR_32)
- `Error` class for Contract ABI, and support of `type="error"` declarations in JSON ABI. (PR_33)
- Error data parsing and matching it with known errors from the ABI when calling `estimate_transact()` and `estimate_deploy()`. (PR_33)

3.6.2 Fixed

- Removed `TxReceipt` export (making an exception here and not counting it as a breaking change, since nobody would have any use for creating one manually). (PR_32)

3.7 0.4.1 (01-05-2022)

3.7.1 Added

- `anyio` support instead of just `trio`. (PR_27)
- Raise `ABIDecodingError` on mismatch between the declared contract ABI and the bytestring returned from `ethCall`. (PR_29)
- Support for gas overrides in `transfer()`, `transact()`, and `deploy()`. (PR_30)

3.8 0.4.0 (23-04-2022)

3.8.1 Changed

- Added type/value checks when normalizing contract arguments. (PR_4)
- Unpacking contract call results into specific types. (PR_4)
- `Address.as_checksum()` renamed to `Address.checksum` (a cached property). (PR_5)
- `ContractABI` and related types reworked. (PR_5)

3.8.2 Added

- Allowed one to declare ABI via Python calls instead of JSON. (PR_4)
- Support for binding of contract arguments to named parameters. (PR_4)
- An `abi.struct()` function to create struct types in contract definitions. (PR_5)
- Hashing, more comparisons and arithmetic functions for `Amount`. (PR_5)
- Hashing and equality for `TxHash`. (PR_5)
- An empty nonpayable constructor is created for a contract if none is specified. (PR_5)
- `RemoteError` and `Unreachable` exception types to report errors from client sessions in a standardized way. (PR_5)

3.9 0.3.0 (03-04-2022)

3.9.1 Changed

- Merged `SigningClient` into `Client`, with the methods of the former now requiring an explicit `Signer` argument. (PR_1)
- Exposed provider sessions via `Client.session()` context manager; all the client methods were moved to the returned session object. (PR_1)

3.9.2 Fixed

- Multiple fixes for typing of methods. (PR_1)
- Fixed the handling of array-of-array ABI types. (PR_2)
- Replaced assertions with more informative exceptions. (PR_3)

3.10 0.2.0 (19-03-2022)

Initial release.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

Z