
py-evm Documentation

Release 0.3.0-alpha.14

Ethereum Foundation

Feb 13, 2020

1	Goals	3
2	Further reading	5
3	Table of contents	7
	Python Module Index	95
	Index	97

Py-EVM is a new implementation of the Ethereum Virtual Machine (EVM) written in Python.

If none of this makes sense to you yet we recommend to checkout the [Ethereum](#) website as well as a [higher level description](#) of the Ethereum project.

CHAPTER 1

Goals

The main focus is to enrich the Ethereum ecosystem with a Python implementation that:

- Supports Ethereum 1.0 as well as 2.0 / Serenity
- Is well documented
- Is easy to understand
- Has clear APIs
- Runs fast and resource friendly
- Is highly flexible to support:
 - Public chains
 - Private chains
 - Consortium chains
 - Advanced research

CHAPTER 2

Further reading

Here are a couple more useful links to check out.

- [Source Code on GitHub](#)
- [Public Gitter Chat](#)
- *[Get involved](#)*

3.1 Introduction

Py-EVM is a new implementation of the Ethereum Virtual Machine (EVM) written in Python.

If none of this makes sense to you yet we recommend to checkout the [Ethereum](#) website as well as a [higher level description](#) of the Ethereum project.

3.1.1 Goals

The main focus is to enrich the Ethereum ecosystem with a Python implementation that:

- Supports Ethereum 1.0 as well as 2.0 / Serenity
- Is well documented
- Is easy to understand
- Has clear APIs
- Runs fast and resource friendly
- Is highly flexible to support:
 - Public chains
 - Private chains
 - Consortium chains
 - Advanced research

3.1.2 Further reading

Here are a couple more useful links to check out.

- [Source Code on GitHub](#)

- [Public Gitter Chat](#)
- [Get involved](#)

3.2 Quickstart

3.2.1 Installation

This guide teaches how to use Py-EVM as a library. For contributors, please check out the [Contributing Guide](#) which explains how to set everything up for development.

Installing on Ubuntu

Py-EVM requires Python 3.6 as well as some tools to compile its dependencies. On Ubuntu, the `python3.6-dev` package contains everything we need. Run the following command to install it.

```
apt-get install python3.6-dev
```

Py-EVM is installed through the pip package manager, if pip isn't available on the system already, we need to install the `python3-pip` package through the following command.

```
apt-get install python3-pip
```

Note: Optional: Often, the best way to guarantee a clean Python 3 environment is with [virtualenv](#). If we don't have `virtualenv` installed already, we first need to install it via pip.

```
pip install virtualenv
```

Then, we can initialize a new virtual environment `venv`, like:

```
virtualenv -p python3 venv
```

This creates a new directory `venv` where packages are installed isolated from any other global packages.

To activate the virtual directory we have to *source* it

```
. venv/bin/activate
```

Finally, we can install the `py-evm` package via pip.

```
pip3 install -U py-evm
```

Installing on macOS

First, install Python 3 with brew:

```
brew install python3
```

Note: Optional: Often, the best way to guarantee a clean Python 3 environment is with [virtualenv](#). If we don't have `virtualenv` installed already, we first need to install it via pip.

```
pip install virtualenv
```

Then, we can initialize a new virtual environment `venv`, like:

```
virtualenv -p python3 venv
```

This creates a new directory `venv` where packages are installed isolated from any other global packages.

To activate the virtual directory we have to *source* it

```
. venv/bin/activate
```

Then, install the `py-evm` package via pip:

```
pip3 install -U py-evm
```

Hint: *Build a first app* on top of Py-EVM in under 5 minutes

3.3 Release notes

3.3.1 py-evm 0.3.0-alpha.14 (2020-02-10)

Features

- Change return type for `import_block` from `Tuple[BlockAPI, Tuple[BlockAPI, ...], Tuple[BlockAPI, ...]]` to `BlockImportResult (NamedTuple)`. (#1910)

Bugfixes

- Fixed a consensus-critical bug for contracts that are created and destroyed in the same block, especially pre-Byzantium. (#1912)

Internal Changes - for Contributors

- Add explicit tests for `validate_header` (#1911)

3.3.2 py-evm 0.3.0-alpha.13 (2020-01-13)

Features

- Make handling of different consensus mechanisms more flexible and sound.
 1. `validate_seal` and `validate_header` are now instance methods. The only reason they can be classmethods today is because our Pow implementation relies on a globally shared cache which should be refactored to use the `ConsensusContextAPI`.

2. There are two new methods: `chain.validate_chain_extension(header, parents)` and `vm.validate_seal_extension`. They perform extension seal checks to support consensus schemes where headers can not be checked if parents are missing.

3. The consensus mechanism is now abstracted via `ConsensusAPI` and `ConsensusContextAPI`. VMs instantiate a consensus api based on the set `consensus_class` and pass it a context which they receive from the chain upon instantiation. The chain instantiates the consensus context api based on the `consensus_context_class`. (#1899)

- Support Istanbul fork in `GOERLI_VM_CONFIGURATION` (#1904)

Bugfixes

- Do not mention PoW in the logging message that we log when `validate_seal` fails. The VM could also be running under a non-PoW consensus mechanism. (#1907)

Deprecations and Removals

- Drop optional `check_seal` param from `VM.validate_header` and turn it into a classmethod. Seal checks now need to be made explicitly via `VM.check_seal` which is also aligned with `VM.check_seal_extension`. (#1909)

3.3.3 py-evm 0.3.0-alpha.12 (2019-12-19)

Features

- Implement the Muir Glacier fork
See: <https://eips.ethereum.org/EIPS/eip-2387> (#1901)

3.3.4 py-evm 0.3.0-alpha.11 (2019-12-12)

Bugfixes

- When double-deleting a storage slot, got `KeyError: (b'\x03', 'key could not be deleted in JournalDB, because it was missing')`. This was fallout from #1893 (#1898)

Performance improvements

- Improve performance when importing a header which is a child of the current canonical chain tip. (#1891)

3.3.5 py-evm 0.3.0-alpha.10 (2019-12-09)

Bugfixes

- Bug: if data was missing during a call to `apply_all_transactions()`, then the call would revert and continue processing transactions. Fix: we re-raise the `EVMMissingData` and do not continue processing transactions. (#1889)
- Fix for net gas metering (EIP-2200) in Istanbul. The “original value” used to calculate gas costs was incorrectly accessing the value at the start of the block, instead of the start of the transaction. (#1893)

Improved Documentation

- Add Matomo Tracking to Docs site.

Matomo is an Open Source web analytics platform that allows us to get better insights and optimize for our audience without the negative consequences of other comparable platforms.

Read more: <https://matomo.org/why-matomo/> (#1892)

3.3.6 py-evm 0.3.0-alpha.9 (2019-12-02)

Features

- Add new Chain APIs (#1887):
 - `get_canonical_block_header_by_number()` (parallel to `get_canonical_block_by_number()`)
 - `get_canonical_transaction_index()`
 - `get_canonical_transaction_by_index()`
 - `get_transaction_receipt_by_index()`

Bugfixes

- Remove the ice age delay that was accidentally left in Istanbul (#1877)

Improved Documentation

- In the API docs display class methods, static methods and methods as one group “methods”. While we ideally wish to separate these, Sphinx keeps them all as one group which we’ll be following until we find a better option. (#794)
- Tweak layout of API docs to improve readability
 - Group API docs by member (methods, attributes) (#1797)
- API doc additions (#1880)
 - Add missing API docs for *MiningChain*.
 - Add missing API docs for `eth.db.*`
 - Add missing API docs for *ConstantinopleVM*, *PetersburgVM* and *IstanbulVM* forks
 - Move all docstrings that aren’t overly specific to a particular implementation from the implementation to the interface. This has the effect that the docstring will appear both on the interface as well as on the implementation except for when the implementation overwrites the docstring with a more specific descriptions.
- Add docstrings to all public APIs that were still lacking one. (#1882)

3.3.7 py-evm 0.3.0-alpha.8 (2019-11-05)

Features

- *Partly* implement Clique consensus according to EIP 225. The implementation doesn't yet cover a mode of operation that would allow to operate as a signer and create blocks. It does however, allow syncing a chain (e.g. Görli) by following the ruleset that is defined in EIP-225. (#1855)
- Set Istanbul block number for mainnet to 9069000, and for Görli to 1561651, as per EIP-1679. (#1858)
- Make the *max length validation* of the *extra_data* field configurable. The reason for that is that different consensus engines such as Clique repurpose this field using different max length limits. (#1864)

Bugfixes

- Resolve version conflict regarding *pluggy* dependency that came up during installation. (#1860)
- Fix issue where Py-EVM crashes when *0* is used as a value for *seal_check_random_sample_rate*. Previously, this would lead to a *DivideByZero* error, whereas now it is recognized as not performing any seal check. This is also symmetric to the current *opposite* behavior of passing *1* to check every single header instead of taking samples. (#1862)
- Improve usability of error message by including hex values of affected hashes. (#1863)
- Gas estimation bugfix: storage values are now correctly reset to original value if the transaction includes a self-destruct, when running estimation iterations. Previously, estimation iterations would produce undefined results, if the transaction included a self-destruct. (#1865)

Performance improvements

- Use new *blake2b-py* library for 560x speedup of Blake2 F compression function. (#1836)

Internal Changes - for Contributors

- Update upstream test fixtures to *v7.0.0 beta.1* and address the two arising disagreements on what accounts should be collected for state trie clearing (as per EIP-161) if a nested call frame had an error. (#1858)

3.3.8 py-evm 0.3.0-alpha.7 (2019-09-19)

Features

- Enable Istanbul fork on Ropsten chain (#1851)

Bugfixes

- Update codebase to more consistently use the `eth_typing.BlockNumber` type. (#1850)

3.3.9 py-evm 0.3.0-alpha.6 (2019-09-05)

Features

- Add EIP-1344 to Istanbul: Chain ID Opcode (#1817)
- Add EIP-152 to Istanbul: Blake2b F Compression precompile at address 9 (#1818)

- Add EIP-2200 to Istanbul: Net gas metering (#1825)
- Add EIP-1884 to Istanbul: Reprice trie-size dependent opcodes (#1826)
- Add EIP-2028: Transaction data gas cost reduction (#1832)
- Expose type hint information via PEP561 (#1845)

Bugfixes

- Add missing `@abstractmethod` decorator to `ConfigurableAPI.configure`. (#1822)

Performance improvements

- ~20% speedup on “simple value transfer” benchmarks, ~10% overall benchmark lift. Optimized retrieval of transactions and receipts from the trie database. (#1841)

Improved Documentation

- Add a “Performance improvements” section to the release notes (#1841)

Deprecations and Removals

- Upgrade to `eth-utils` $\geq 1.7.0$ which removes the `eth.tools.logging` module implementations of `ExtendedDebugLogger` in favor of the ones exposed by the `eth-utils` library. This also removes the automatic setup of the `DEBUG2` logging level which was previously a side effect of importing the `eth` module. See `eth_utils.setup_DEBUG2_logging` for more information. (#1846)

3.3.10 py-evm 0.3.0-alpha.5 (2019-08-22)

Features

- Add EIP-1108 to Istanbul: Reduce EC precompile costs (#1819)

Bugfixes

- Make sure `persist_checkpoint_header` sets the given header as canonical head. (#1830)

Improved Documentation

- Remove section on Trinity’s goals from the Readme. It’s been a leftover from when Py-EVM and Trinity where hosted in a single repository. (#1827)

3.3.11 py-evm 0.3.0-alpha.4 (2019-08-19)

Features

- Add an *optional* `genesis_parent_hash` parameter to `persist_header_chain()` and `persist_block()` that allows to overwrite the hash that is used to identify the genesis header. This allows persisting headers / blocks that aren't (yet) connected back to the true genesis header.

This feature opens up new, faster syncing techniques. (#1823)

Bugfixes

- Add missing `@abstractmethod` decorator to `ConfigurableAPI.configure`. (#1822)

Deprecations and Removals

- Remove `AsyncHeaderDB` that wasn't used anywhere (#1823)

3.3.12 py-evm 0.3.0-alpha.3 (2019-08-13)

Bugfixes

- Add back missing `Chain.get_vm_class` method. (#1821)

3.3.13 py-evm 0.3.0-alpha.2 (2019-08-13)

Features

- Package up test suites for the `DatabaseAPI` and `AtomicDatabaseAPI` to be class-based to make them reusable by other libraries. (#1813)

Bugfixes

- Fix a crash during chain reorganization on a header-only chain (which can happen during Beam Sync) (#1810)

Improved Documentation

- Setup towncrier to generate release notes from fragment files to ensure a higher standard for release notes. (#1796)

Deprecations and Removals

- Drop `StateRootNotFound` as an over-specialized version of `EVMMissingData`. Drop `VMState.execute_transaction()` as redundant to `VMState.apply_transaction()`. (#1809)

3.3.14 v0.3.0-alpha.1

Released 2019-06-05 (off-schedule release to handle eth-keys dependency issue)

- #1785: Breaking Change: Dropped python3.5 support
- #1788: Fix dependency issue with eth-keys, don't allow v0.3+ for now

3.3.15 0.2.0-alpha.43

Released 2019-05-20

- #1778: Feature: Raise custom decorated exceptions when a trie node is missing from the database (plus some bonus logging and performance improvements)
- #1732: Bugfix: squashed an occasional "mix hash mismatch" while syncing
- #1716: Performance: only calculate & persist state root at end of block (post-Byzantium)
- #1735:
 - Performance: only calculate & persist storage roots at end of block (post-Byzantium)
 - Performance: batch all account trie writes to the database once per block
- #1747:
 - Maintenance: Lazily generate VM.block on first access. Enables loading the VM when you don't have its block body.
 - Performance: Fewer DB reads when block is never accessed.
- Performance: speedups on `chain.import_block()`:
 - #1764: Speed up `is_valid_opcode` check, formerly 7% of total import time! (now less than 1%)
 - #1765: Reduce logging overhead, ~15% speedup
 - #1766: Cache transaction sender, ~3% speedup
 - #1770: Faster bytecode iteration, ~2.5% speedup
 - #1771: Faster opcode lookup in `apply_computation`, ~1.5% speedup
 - #1772: Faster Journal access of latest data, ~6% speedup
 - #1773: Faster stack operations, ~9% speedup
 - #1776: Faster Journal record & commit checkpoints, ~7% speedup
 - #1777: Faster bytecode navigation, ~7% speedup
- #1751: Maintenance: Add placeholder for Istanbul fork

3.3.16 0.2.0-alpha.42

Released 2019-02-28

- #1719: Implement and activate Petersburg fork (aka Constantinople fixed)
- #1718: Performance: faster account lookups in EVM
- #1670: Performance: lazily look up ancestor block hashes, and cache result, so looking up parent hash in EVM is faster than `grand^100` parent

3.3.17 0.2.0-alpha.40

Released Jan 15, 2019

- #1717: Indefinitely postpone the pending Constantinople release
- #1715: Remove Eth2 Beacon code, moving to trinity project

3.4 Cookbook

The Cookbook is a collection of simple recipes that demonstrate good practices to accomplish common tasks. The examples are usually short answers to simple “How do I...” questions that go beyond simple API descriptions but also don’t need a full guide to become clear.

3.4.1 Using the Chain object

A “single” blockchain is made by a series of different virtual machines for different spans of blocks. For example, the Ethereum mainnet had one virtual machine for blocks 0 till 1150000 (known as Frontier), and another VM for blocks 1150000 till 1920000 (known as Homestead).

The Chain object manages the series of fork rules, after you define the VM ranges. For example, to set up a chain that would track the mainnet Ethereum network until block 1920000, you could create this chain class:

```
>>> from eth import constants, Chain
>>> from eth.vm.forks.frontier import FrontierVM
>>> from eth.vm.forks.homestead import HomesteadVM
>>> from eth.chains.mainnet import HOMESTEAD_MAINNET_BLOCK

>>> chain_class = Chain.configure(
...     __name__='Test Chain',
...     vm_configuration=(
...         (constants.GENESIS_BLOCK_NUMBER, FrontierVM),
...         (HOMESTEAD_MAINNET_BLOCK, HomesteadVM),
...     ),
... )
```

Then to initialize, you can start it up with an in-memory database:

```
>>> from eth.db.atomic import AtomicDB
>>> from eth.chains.mainnet import MAINNET_GENESIS_HEADER

>>> # start a fresh in-memory db

>>> # initialize a fresh chain
>>> chain = chain_class.from_genesis_header(AtomicDB(), MAINNET_GENESIS_HEADER)
```

3.4.2 Creating a chain with custom state

While the previous recipe demos how to create a chain from an existing genesis header, we can also create chains simply by specifying various genesis parameter as well as an optional genesis state.

```

>>> from eth_keys import keys
>>> from eth import constants
>>> from eth.chains.mainnet import MainnetChain
>>> from eth.db.atomic import AtomicDB
>>> from eth_utils import to_wei, encode_hex

>>> # Giving funds to some address
>>> SOME_ADDRESS = b'\x85\xa2\x89V\xb9%\x93M\x03\xdd\xb4Xu\xe1\x8e\x85\x93\x12\xc1
↳ '
>>> GENESIS_STATE = {
...     SOME_ADDRESS: {
...         "balance": to_wei(10000, 'ether'),
...         "nonce": 0,
...         "code": b'',
...         "storage": {}
...     }
... }

>>> GENESIS_PARAMS = {
...     'parent_hash': constants.GENESIS_PARENT_HASH,
...     'uncles_hash': constants.EMPTY_UNCLE_HASH,
...     'coinbase': constants.ZERO_ADDRESS,
...     'transaction_root': constants.BLANK_ROOT_HASH,
...     'receipt_root': constants.BLANK_ROOT_HASH,
...     'difficulty': constants.GENESIS_DIFFICULTY,
...     'block_number': constants.GENESIS_BLOCK_NUMBER,
...     'gas_limit': constants.GENESIS_GAS_LIMIT,
...     'extra_data': constants.GENESIS_EXTRA_DATA,
...     'nonce': constants.GENESIS_NONCE
... }

>>> chain = MainnetChain.from_genesis(AtomicDB(), GENESIS_PARAMS, GENESIS_STATE)

```

3.4.3 Getting the balance from an account

Considering our previous example, we can get the balance of our pre-funded account as follows.

```

>>> current_vm = chain.get_vm()
>>> state = current_vm.state
>>> state.get_balance(SOME_ADDRESS)
1000000000000000000000000

```

3.4.4 Building blocks incrementally

The default Chain is stateless and thus does not keep a tip block open that would allow us to incrementally build a block. However, we can import the MiningChain which does allow exactly that.

```

>>> from eth.chains.base import MiningChain

```

Please check out the *Understanding the mining process* guide for a full example that demonstrates how to use the MiningChain.

3.5 Guides

This section aims to provide hands-on guides to demonstrate how to use Py-EVM. If you are looking for detailed API descriptions check out the [API section](#).

3.5.1 Building an app that uses Py-EVM

One of the primary use cases of the Py-EVM library is to enable developers to build applications that want to interact with the ethereum ecosystem.

In this guide we want to build a very simple script that uses the Py-EVM library to create a fresh blockchain with a pre-funded address to simply read the balance of that address through the regular Py-EVM APIs. Frankly, not the most exciting application in the world, but the principle of how we use the Py-EVM library stays the same for more exciting use cases.

Setting up the application

Let's get started by setting up a new application. Often, that process involves lots of repetitive boilerplate code, so instead of doing it all by hand, let's just clone the [Ethereum Python Project Template](#) which contains all the typical things that we want.

To clone this into a new directory `demo-app` run:

```
git clone https://github.com/carver/ethereum-python-project-template.git demo-app
```

Then, change into the directory

```
cd demo-app
```

Add the Py-EVM library as a dependency

To add Py-EVM as a dependency, open the `setup.py` file in the root directory of the application and change the `install_requires` section as follows.

```
install_requires=[
    "eth-utils>=1,<2",
    "py-vm==0.2.0a40",
],
```

Warning: Make sure to also change the name inside the `setup.py` file to something valid (e.g. `demo-app`) or otherwise, fetching dependencies will fail.

Next, we need to use the `pip` package manager to fetch and install the dependencies of our app.

Note: Optional: Often, the best way to guarantee a clean Python 3 environment is with [virtualenv](#). If we don't have `virtualenv` installed already, we first need to install it via `pip`.

```
pip install virtualenv
```

Then, we can initialize a new virtual environment `venv`, like:

```
virtualenv -p python3 venv
```

This creates a new directory `venv` where packages are installed isolated from any other global packages.

To activate the virtual directory we have to *source* it

```
. venv/bin/activate
```

To install the dependencies, run:

```
pip install -e .[dev]
```

Congrats! We're now ready to build our application!

Writing the application code

Next, we'll create a new directory `app` and create a file `main.py` inside. Paste in the following content.

Note: The code examples are often written in an interactive session syntax, which is indicated by lines beginning with `>>>` or `...` . This enables us to run automatic tests against the examples to ensure they keep working while the library is evolving. When we want to copy and paste example code to play with it, we need to remove these extra characters to get runnable valid Python code.

```
>>> from eth import constants
>>> from eth.chains.mainnet import MainnetChain
>>> from eth.db.atomic import AtomicDB

>>> from eth_utils import to_wei, encode_hex

>>> MOCK_ADDRESS = constants.ZERO_ADDRESS
>>> DEFAULT_INITIAL_BALANCE = to_wei(10000, 'ether')

>>> GENESIS_PARAMS = {
...     'parent_hash': constants.GENESIS_PARENT_HASH,
...     'uncles_hash': constants.EMPTY_UNCLE_HASH,
...     'coinbase': constants.ZERO_ADDRESS,
...     'transaction_root': constants.BLANK_ROOT_HASH,
...     'receipt_root': constants.BLANK_ROOT_HASH,
...     'difficulty': constants.GENESIS_DIFFICULTY,
...     'block_number': constants.GENESIS_BLOCK_NUMBER,
...     'gas_limit': constants.GENESIS_GAS_LIMIT,
...     'extra_data': constants.GENESIS_EXTRA_DATA,
...     'nonce': constants.GENESIS_NONCE
... }

>>> GENESIS_STATE = {
...     MOCK_ADDRESS: {
...         "balance": DEFAULT_INITIAL_BALANCE,
...         "nonce": 0,
...         "code": b'',
...         "storage": {}
...     }
... }
```

(continues on next page)

(continued from previous page)

```

... }
>>> chain = MainnetChain.from_genesis(AtomicDB(), GENESIS_PARAMS, GENESIS_STATE)
>>> mock_address_balance = chain.get_vm().state.get_balance(MOCK_ADDRESS)
>>> print("The balance of address {} is {} wei".format(
...     encode_hex(MOCK_ADDRESS),
...     mock_address_balance)
... )
The balance of address 0x0000000000000000000000000000000000000000000000000000000000000000 is_
↪1000000000000000000000000000000000 wei

```

Runing the script

Let's run the script by invoking the following command.

```
python app/main.py
```

We should see the following output.

```

The balance of address 0x0000000000000000000000000000000000000000000000000000000000000000 is_
↪1000000000000000000000000000000000 wei

```

3.5.2 Architecture

The primary use case for Py-EVM is supporting the public Ethereum blockchain.

However, it is architected with a strong focus on configurability and extensibility. Use of Py-EVM for alternate use cases such as private chains, consortium chains, or even chains with fundamentally different VM semantics should be possible without any changes to the core library.

The following abstractions are used to represent the full consensus rules for a Py-EVM based blockchain.

- Chain: High level API for interacting with the blockchain.
- VM: High level API for a single fork within a Chain
- VMState: The current state of the VM, transaction execution logic and the state transition function.
- Message: Representation of the portion of the transaction which is relevant to VM execution.
- Computation: The computational state and result of VM execution.
- Opcode: The logic for a single opcode.

The Chain

The term **Chain** is used to encapsulate:

- The state transition function (e.g. VM opcodes and execution logic)
- Protocol rules (e.g. block rewards, header rewards, difficulty calculations, transaction execution)
- The chain data (e.g. **Headers**, **Blocks**, **Transactions** and **Receipts**)
- The state data (e.g. **balance**, **nonce**, **code** and **storage**)

- The chain state (e.g. tracking the chain head, canonical blocks)

Note: While a chain is used to *wrap* these concepts, many of them are actually defined at lower layers such as the underlying **Virtual Machines**.

The `Chain` object itself is largely an interface and orchestration layer. Most of the `Chain` APIs merely serving as a passthrough to the appropriate `VM`.

A chain has one or more underlying **Virtual Machines** or `VMs`. The chain contains a mapping which defines which `VM` should be active for which blocks.

The chain for the public mainnet Ethereum blockchain would have a separate `VM` defined for each fork ruleset (e.g. **Frontier**, **Homestead**, **Tangerine Whistle**, **Spurious Dragon**, **Byzantium**).

The VM

The term **VM** is used to encapsulate:

- The state transition function for a single fork ruleset.
- Orchestration logic for transaction execution.
- Block construction and validation.
- Chain data storage and retrieval APIs

The `VM` object loosely mirrors many of the `Chain` APIs for retrieval of chain state such as blocks, headers, transactions and receipts. It is also responsible for block level protocol logic such as block creation and validation.

The VMState

The term **VMState** is used to encapsulate:

- Execution context for the `VM` (e.g. `coinbase` or `gas_limit`)
- The state root defining the current `VM` state.
- Some block validation

The Message

The term **Message** comes from the yellow paper. It encapsulates the information from the transaction needed to initiate the outermost layer of `VM` execution.

- Parameters like `sender`, `value`, `to`

The message can be thought of as the `VM`'s internal representation of a transaction.

The Computation

The term **Computation** is used to encapsulate:

- The computational state during `VM` execution (e.g. `memory`, `stack`, `gas metering`)
- The computational results of `VM` execution (e.g. `return data`, `gas consumption` and `refunds`, `execution errors`)

This abstraction is the interface through which opcode logic is implemented.

The Opcode

The term **Opcode** is used to encapsulate:

- A single instruction within the VM such as the `ADD` or `MUL` opcodes.

Opcodes are implemented as `TODO`

3.5.3 Understanding the mining process

From the *Cookbook* we can already learn how to use the *Chain* class to create a single blockchain as a combination of different virtual machines for different spans of blocks.

In this guide we want to build up on that knowledge and look into the actual mining process.

Note: Mining is an overloaded term and in fact the names of the mentioned APIs are subject to change.

Mining

The term *mining* can refer to different things depending on our point of view. Most of the time when we read about *mining*, we talk about the process where several parties are *competing* to be the first to create a new valid block and pass it on to the network.

In this guide, when we talk about the `mine_block()` API, we are only referring to the part that creates, validates and sets a block as the new canonical head of the chain but not necessarily as part of the mentioned competition to be the first. In fact, the `mine_block()` API is internally also called when we import existing blocks that others created.

Mining an empty block

Usually when we think about creating blocks we naturally think about adding transactions to the block first because, after all, one primary use case for the Ethereum blockchain is to process *transactions* which are wrapped in blocks.

For the sake of simplicity though, we'll mine an empty block as a first example (meaning the block will not contain any transactions)

As a refresher, here's how we create a chain as demonstrated in the *Using the chain object recipe* from the cookbook.

```
from eth.db.atomic import AtomicDB
from eth.chains.mainnet import MAINNET_GENESIS_HEADER

# increase the gas limit
genesis_header = MAINNET_GENESIS_HEADER.copy(gas_limit=3141592)

# initialize a fresh chain
chain = chain_class.from_genesis_header(AtomicDB(), genesis_header)
```

Since we decided to not add any transactions to our block let's just call `mine_block()` and see what happens.

```
# initialize a fresh chain
chain = chain_class.from_genesis_header(AtomicDB(), genesis_header)

chain.mine_block()
```

Aw, snap! We're running into an exception at `check_pow()`. Apparently we are trying to add a block to the chain that doesn't qualify the Proof-of-Work (PoW) rules. The error tells us precisely that the `mix_hash` of our block does not match the expected value.

```
Traceback (most recent call last):
  File "scripts/benchmark/run.py", line 111, in <module>
    run()
  File "scripts/benchmark/run.py", line 52, in run
    block = chain.mine_block() ***pow_args
  File "/py-evm/eth/chains/base.py", line 545, in mine_block
    self.validate_block(mined_block)
  File "/py-evm/eth/chains/base.py", line 585, in validate_block
    self.validate_seal(block.header)
  File "/py-evm/eth/chains/base.py", line 622, in validate_seal
    header.mix_hash, header.nonce, header.difficulty)
  File "/py-evm/eth/consensus/pow.py", line 70, in check_pow
    encode_hex(mining_output[b'mix digest']), encode_hex(mix_hash))

eth.exceptions.ValidationError: mix hash mismatch;
0x7a76bbf0c8d0e683fafa2d7cab27f601e19f35e7ecad7e1abb064b6f8f08fe21 !=
0x0000000000000000000000000000000000000000000000000000000000000000
```

Let's lookup how `check_pow()` is implemented.

```
def check_pow(block_number: int,
               mining_hash: Hash32,
               mix_hash: Hash32,
               nonce: bytes,
               difficulty: int) -> None:
    validate_length(mix_hash, 32, title="Mix Hash")
    validate_length(mining_hash, 32, title="Mining Hash")
    validate_length(nonce, 8, title="POW Nonce")
    cache = get_cache(block_number)
    mining_output = hashimoto_light(
        block_number, cache, mining_hash, big_endian_to_int(nonce))
    if mining_output[b'mix digest'] != mix_hash:
        raise ValidationError(
            f"mix hash mismatch; expected: {encode_hex(mining_output[b'mix digest'])}
↪      f"!= actual: {encode_hex(mix_hash)}. "
            f"Mix hash calculated from block #{block_number}, "
            f"mine hash {encode_hex(mining_hash)}, nonce {encode_hex(nonce)}"
            f", difficulty {difficulty}, cache hash {encode_hex(keccak(cache))}"
        )
    result = big_endian_to_int(mining_output[b'result'])
    validate_lte(result, 2**256 // difficulty, title="POW Difficulty")
```

Just by looking at the signature of that function we can see that validating the PoW is based on the following parameters:

- `block_number` - the number of the given block
- `difficulty` - the difficulty of the PoW algorithm
- `mining_hash` - hash of the mining header
- `mix_hash` - together with the nonce forms the actual proof
- `nonce` - together with the `mix_hash` forms the actual proof

The PoW algorithm checks that all these parameters match correctly, ensuring that only valid blocks can be added to the chain.

In order to produce a valid block, we have to set the correct `mix_hash` and `nonce` in the header. We can pass these as key-value pairs when we call `mine_block()` as seen below.

```
chain.mine_block(nonce=valid_nonce, mix_hash=valid_mix_hash)
```

This call will work just fine assuming we are passing the correct `nonce` and `mix_hash` that corresponds to the block getting mined.

Retrieving a valid nonce and mix hash

Now that we know we can call `mine_block()` with the correct parameters to successfully add a block to our chain, let's briefly go over an example that demonstrates how we can retrieve a matching `nonce` and `mix_hash`.

Note: Py-EVM currently doesn't offer a stable API for actual PoW mining. The following code is for demonstration purpose only.

Mining on the main ethereum chain is a competition done simultaneously by many miners, hence the *mining difficulty* is pretty high which means it will take a very long time to find the right `nonce` and `mix_hash` on commodity hardware. In order for us to have something that we can tinker with on a regular laptop, we'll construct a test chain with the `difficulty` set to 1.

Let's start off by defining the `GENESIS_PARAMS`.

```
from eth import constants

GENESIS_PARAMS = {
    'parent_hash': constants.GENESIS_PARENT_HASH,
    'uncles_hash': constants.EMPTY_UNCLE_HASH,
    'coinbase': constants.ZERO_ADDRESS,
    'transaction_root': constants.BLANK_ROOT_HASH,
    'receipt_root': constants.BLANK_ROOT_HASH,
    'difficulty': 1,
    'block_number': constants.GENESIS_BLOCK_NUMBER,
    'gas_limit': 3141592,
    'timestamp': 1514764800,
    'extra_data': constants.GENESIS_EXTRA_DATA,
    'nonce': constants.GENESIS_NONCE
}
```

Next, we'll create the chain itself using the defined `GENESIS_PARAMS` and the latest `ByzantiumVM`.

```
from eth import MiningChain
from eth.vm.forks.byzantium import ByzantiumVM
from eth.db.backends.memory import AtomicDB

klass = MiningChain.configure(
    __name__='TestChain',
    vm_configuration=(
        constants.GENESIS_BLOCK_NUMBER, ByzantiumVM,
    ))
chain = klass.from_genesis(AtomicDB(), GENESIS_PARAMS)
```

Now that we have the building blocks available, let's put it all together and mine a proper block!

```
from eth.consensus.pow import mine_pow_nonce

# We have to finalize the block first in order to be able read the
# attributes that are important for the PoW algorithm
block = chain.get_vm().finalize_block(chain.get_block())

# based on mining_hash, block number and difficulty we can perform
# the actual Proof of Work (PoW) mechanism to mine the correct
# nonce and mix_hash for this block
nonce, mix_hash = mine_pow_nonce(
    block.number,
    block.header.mining_hash,
    block.header.difficulty)

block = chain.mine_block(mix_hash=mix_hash, nonce=nonce)
```

```
>>> print(block)
Block #1
```

Let's take a moment to fully understand what this code does.

1. We call `finalize_block()` on the underlying VM in order to retrieve the information that we need to calculate the nonce and the `mix_hash`.
2. We then call `mine_pow_nonce()` to retrieve the proper nonce and `mix_hash` that we need to mine the block and satisfy the validation.
3. Finally we call `mine_block()` and pass along the nonce and the `mix_hash`

Note: The code above will essentially perform `finalize_block` twice. Keep in mind this code is for demonstration purpose only and that Py-EVM will provide a pluggable system in the future to allow PoW mining among other things.

Mining a block with transactions

Now that we've learned the basics of how the mining process works, let's revisit our example and add a transaction before we mine another block. There are a couple of concepts we need to dive into in order to accomplish that goal.

Every transaction goes from a sender `Address` to a receiver `Address`. Each transaction takes some computational power to get processed that is measured in a unit called `gas`.

In practice, we have to pay the miners to put our transaction in a block. However, there is no *technical* reason why we have to pay for the computing power, but only an economical, i.e. in reality we'll usually have trouble finding a miner who's willing to include a transaction that doesn't pay for its computational costs.

In this example, however, **we are the miner** which means we are free to include any transactions we like. In the spirit of this guide, let's start simple and create a transaction that sends zero ether from one address to another address. Keep in mind that even if the value being transferred is zero, there's still a computational cost for the processing but since we are the miner, we'll mine it anyway even if no one is willing to pay for it!

Let's first setup the sender and receiver.

(continued from previous page)

```

>>> # We have to finalize the block first in order to be able read the
>>> # attributes that are important for the PoW algorithm
>>> block = chain.get_vm().finalize_block(chain.get_block())

>>> # based on mining_hash, block number and difficulty we can perform
>>> # the actual Proof of Work (PoW) mechanism to mine the correct
>>> # nonce and mix_hash for this block
>>> nonce, mix_hash = mine_pow_nonce(
...     block.number,
...     block.header.mining_hash,
...     block.header.difficulty
... )

>>> chain.mine_block(mix_hash=mix_hash, nonce=nonce)
<ByzantiumBlock(#Block #1)>

```

3.5.4 Creating Opcodes

An opcode is just a function which takes a *BaseComputation* instance as it's sole argument. If an opcode function has a return value, this value will be discarded during normal VM execution.

Here are some simple examples.

```

def noop(computation):
    """
    An opcode which does nothing (not even consume gas)
    """
    pass

def burn_5_gas(computation):
    """
    An opcode which simply burns 5 gas
    """
    computation.consume_gas(5, reason='why not?')

```

The `as_opcode()` helper

While these examples are demonstrative of *simple* logic, opcodes will traditionally have an intrinsic gas cost associated with them. Py-EVM offers an abstraction which allows for decoupling of gas consumption from opcode logic which can be convenient for cases where an opcode's gas cost changes between different VM rules but its logic remains constant.

`eth.vm.opcode.as_opcode(logic_fn, mnemonic, gas_cost)`

- The `logic_fn` argument should be a callable conforming to the opcode API, taking a `~eth.vm.computation.Computation` instance as its sole argument.
- The `mnemonic` is a string such as 'ADD' or 'MUL'.
- The `gas_cost` is the gas cost to execute this opcode.

The return value is a function which will consume the `gas_cost` prior to execution of the `logic_fn`.

Usage of the `as_opcode()` helper:

```
def custom_op(computation):
    ... # opcode logic here

class ExampleComputation(BaseComputation):
    opcodes = {
        b'\x01': as_opcode(custom_op, 'CUSTOM_OP', 10),
    }
```

Opcodes as classes

Sometimes it may be helpful to share common logic between similar opcodes, or the same opcode across multiple fork rules. In these cases, implementing opcodes as classes *may* be the right choice. This is as simple as implementing a `__call__` method on your class which conforms to the opcode API, taking a single `Computation` instance as the sole argument.

```
class MyOpcode:
    def initial_logic(self, computation):
        ...

    def main_logic(self, computation):
        ...

    def cleanup_logic(self, computation):
        ...

    def __call__(self, computation):
        self.initial_logic(computation)
        self.main_logic(computation)
        self.cleanup_logic(computation)
```

With this pattern, the overall structure, as well as much of the logic can be re-used while still allowing a mechanism for overriding individual sections of the opcode logic.

3.6 API

This section aims to provide a detailed description of all APIs. If you are looking for something more hands-on or higher-level check out the existing *guides*.

Warning: We expect each alpha release to have breaking changes to the API.

3.6.1 ABC

Abstract base classes for documented interfaces

MiningHeaderAPI

```
class eth.abc.MiningHeaderAPI(*args, **kwargs)
```

A class to define a block header without `mix_hash` and `nonce` which can act as a temporary representation during mining before the block header is sealed.

BlockHeaderAPI

class `eth.abc.BlockHeaderAPI (*args, **kwargs)`

A class derived from `MiningHeaderAPI` to define a block header after it is sealed.

LogAPI

class `eth.abc.LogAPI (*args, **kwargs)`

A class to define a written log.

ReceiptAPI

class `eth.abc.ReceiptAPI (*args, **kwargs)`

A class to define a receipt to capture the outcome of a transaction.

BaseTransactionAPI

class `eth.abc.BaseTransactionAPI`

A class to define all common methods of a transaction.

copy (***overrides*) → T

Return a copy of the transaction.

gas_used_by (*computation: eth.abc.ComputationAPI*) → int

Return the gas used by the given computation. In Frontier, for example, this is sum of the intrinsic cost and the gas used during computation.

get_intrinsic_gas () → int

Return the intrinsic gas for the transaction which is defined as the amount of gas that is needed before any code runs.

validate () → None

Hook called during instantiation to ensure that all transaction parameters pass validation rules.

intrinsic_gas

Convenience property for the return value of `get_intrinsic_gas`

TransactionFieldsAPI

class `eth.abc.TransactionFieldsAPI`

A class to define all common transaction fields.

UnsignedTransactionAPI

class `eth.abc.UnsignedTransactionAPI (*args, **kwargs)`

A class representing a transaction before it is signed.

as_signed_transaction (*private_key: eth_keys.datatypes.PrivateKey*) → `eth.abc.SignedTransactionAPI`

Return a version of this transaction which has been signed using the provided *private_key*

SignedTransactionAPI

class `eth.abc.SignedTransactionAPI (*args, **kwargs)`
 A class representing a transaction that was signed with a private key.

check_signature_validity () → None
 Check if the signature is valid. Raise a `ValidationError` if the signature is invalid.

classmethod **create_unsigned_transaction** (*, *nonce: int, gas_price: int, gas: int, to: NewType.<locals>.new_type, value: int, data: bytes*) → `eth.abc.UnsignedTransactionAPI`
 Create an unsigned transaction.

classmethod **from_base_transaction** (*transaction: eth.abc.SignedTransactionAPI*) → `eth.abc.SignedTransactionAPI`
 Create a signed transaction from a base transaction.

get_message_for_signing () → bytes
 Return the bytestring that should be signed in order to create a signed transactions

get_sender () → `NewType.<locals>.new_type`
 Get the 20-byte address which sent this transaction.
 This can be a slow operation. `transaction.sender` is always preferred.

validate () → None
 Hook called during instantiation to ensure that all transaction parameters pass validation rules.

is_signature_valid
 Return `True` if the signature is valid, otherwise `False`.

sender
 Convenience and performance property for the return value of `get_sender`

BlockAPI

class `eth.abc.BlockAPI (*args, **kwargs)`
 A class to define a block.

classmethod **from_header** (*header: eth.abc.BlockHeaderAPI, chaindb: eth.abc.ChainDatabaseAPI*) → `eth.abc.BlockAPI`
 Instantiate a block from the given `header` and the `chaindb`.

classmethod **get_transaction_class** () → `Type[eth.abc.SignedTransactionAPI]`
 Return the transaction class that is valid for the block.

hash
 Return the hash of the block.

is_genesis
 Return `True` if this block represents the genesis block of the chain, otherwise `False`.

number
 Return the number of the block.

DatabaseAPI

class `eth.abc.DatabaseAPI`
 A class representing a database.

delete (*key: bytes*) → None
 Delete the given key from the database.

exists (*key: bytes*) → bool
 Return True if the key exists in the database, otherwise False.

set (*key: bytes, value: bytes*) → None
 Assign the value to the key.

AtomicDatabaseAPI

class eth.abc.AtomicDatabaseAPI
 Like BatchDB, but immediately write out changes if they are not in an `atomic_batch()` context.

atomic_batch () → ContextManager[eth.abc.DatabaseAPI]
 Return a `ContextManager` to write an atomic batch to the database.

HeaderDatabaseAPI

class eth.abc.HeaderDatabaseAPI (*db: eth.abc.AtomicDatabaseAPI*)
 A class representing a database for block headers.

get_block_header_by_hash (*block_hash: NewType.<locals>.new_type*) → eth.abc.BlockHeaderAPI
 Return the block header for the given `block_hash`. Raise `HeaderNotFound` if no header with the given `block_hash` exists in the database.

get_canonical_block_hash (*block_number: NewType.<locals>.new_type*) → NewType.<locals>.new_type
 Return the block hash for the canonical block at the given number.
 Raise `BlockNotFound` if there's no block header with the given number in the canonical chain.

get_canonical_block_header_by_number (*block_number: NewType.<locals>.new_type*) → eth.abc.BlockHeaderAPI
 Return the block header with the given number in the canonical chain.
 Raise `HeaderNotFound` if there's no block header with the given number in the canonical chain.

get_canonical_head () → eth.abc.BlockHeaderAPI
 Return the current block header at the head of the chain.

get_score (*block_hash: NewType.<locals>.new_type*) → int
 Return the score for the given `block_hash`.

header_exists (*block_hash: NewType.<locals>.new_type*) → bool
 Return True if the `block_hash` exists in the database, otherwise False.

persist_checkpoint_header (*header: eth.abc.BlockHeaderAPI, score: int*) → None
 Persist a checkpoint header with a trusted score. Persisting the checkpoint header automatically sets it as the new canonical head.

persist_header (*header: eth.abc.BlockHeaderAPI*) → Tuple[Tuple[eth.abc.BlockHeaderAPI, ...], Tuple[eth.abc.BlockHeaderAPI, ...]]
 Persist the header in the database. Return two iterable of headers, the first containing the new canonical header, the second containing the old canonical headers

persist_header_chain (*headers*: Sequence[eth.abc.BlockHeaderAPI], *genesis_parent_hash*: NewType.<locals>.new_type = None) → Tuple[Tuple[eth.abc.BlockHeaderAPI, ...], Tuple[eth.abc.BlockHeaderAPI, ...]]

Persist a chain of headers in the database. Return two iterable of headers, the first containing the new canonical headers, the second containing the old canonical headers

Parameters **genesis_parent_hash** – optional parent hash of the block that is treated as genesis. Providing a `genesis_parent_hash` allows storage of headers that aren't (yet) connected back to the true genesis header.

ChainDatabaseAPI

class eth.abc.ChainDatabaseAPI (*db*: eth.abc.AtomicDatabaseAPI)

A class representing a database for chain data. This class is derived from `HeaderDatabaseAPI`.

add_receipt (*block_header*: eth.abc.BlockHeaderAPI, *index_key*: int, *receipt*: eth.abc.ReceiptAPI) → NewType.<locals>.new_type

Add the given receipt to the provided block header.

Return the updated `receipts_root` for updated block header.

add_transaction (*block_header*: eth.abc.BlockHeaderAPI, *index_key*: int, *transaction*: eth.abc.SignedTransactionAPI) → NewType.<locals>.new_type

Add the given transaction to the provided block header.

Return the updated `transactions_root` for updated block header.

exists (*key*: bytes) → bool

Return `True` if the given key exists in the database.

get (*key*: bytes) → bytes

Return the value for the given key or a `KeyError` if it doesn't exist in the database.

get_block_transaction_hashes (*block_header*: eth.abc.BlockHeaderAPI) → Tuple[NewType.<locals>.new_type, ...]

Return a tuple containing the hashes of the transactions of the given `block_header`.

get_block_transactions (*block_header*: eth.abc.BlockHeaderAPI, *transaction_class*: Type[eth.abc.SignedTransactionAPI]) → Tuple[eth.abc.SignedTransactionAPI, ...]

Return an iterable of transactions for the block specified by the given block header.

get_block_uncles (*uncles_hash*: NewType.<locals>.new_type) → Tuple[eth.abc.BlockHeaderAPI, ...]

Return an iterable of uncle headers specified by the given `uncles_hash`

get_receipt_by_index (*block_number*: NewType.<locals>.new_type, *receipt_index*: int) → eth.abc.ReceiptAPI

Return the receipt of the transaction at specified index for the block header obtained by the specified block number

get_receipts (*header*: eth.abc.BlockHeaderAPI, *receipt_class*: Type[eth.abc.ReceiptAPI]) → Tuple[eth.abc.ReceiptAPI, ...]

Return a tuple of receipts for the block specified by the given block header.

get_transaction_by_index (*block_number*: NewType.<locals>.new_type, *transaction_index*: int, *transaction_class*: Type[eth.abc.SignedTransactionAPI]) → eth.abc.SignedTransactionAPI

Return the transaction at the specified `transaction_index` from the block specified by `block_number` from the canonical chain.

Raise `TransactionNotFound` if no block with that `block_number` exists.

get_transaction_index (*transaction_hash*: `NewType.<locals>.new_type`) → `Tuple[NewType.<locals>.new_type, int]`

Return a 2-tuple of (`block_number`, `transaction_index`) indicating which block the given transaction can be found in and at what index in the block transactions.

Raise `TransactionNotFound` if the `transaction_hash` is not found in the canonical chain.

persist_block (*block*: `eth.abc.BlockAPI`, *genesis_parent_hash*: `NewType.<locals>.new_type` = `None`) → `Tuple[Tuple[NewType.<locals>.new_type, ...], Tuple[NewType.<locals>.new_type, ...]]`

Persist the given block's header and uncles.

Parameters

- **block** – the block that gets persisted
- **genesis_parent_hash** – *optional* parent hash of the header that is treated as genesis. Providing a `genesis_parent_hash` allows storage of blocks that aren't (yet) connected back to the true genesis header.

Assumes all block transactions have been persisted already.

persist_trie_data_dict (*trie_data_dict*: `Dict[NewType.<locals>.new_type, bytes]`) → `None`

Store raw trie data to db from a dict

persist_uncles (*uncles*: `Tuple[eth.abc.BlockHeaderAPI]`) → `NewType.<locals>.new_type`

Persist the list of uncles to the database.

Return the uncles hash.

GasMeterAPI

class `eth.abc.GasMeterAPI`

A class to define a gas meter.

consume_gas (*amount*: `int`, *reason*: `str`) → `None`

Consume amount of gas for a defined reason.

refund_gas (*amount*: `int`) → `None`

Refund amount of gas.

return_gas (*amount*: `int`) → `None`

Return amount of gas.

MessageAPI

class `eth.abc.MessageAPI`

A message for VM computation.

OpcodeAPI

class `eth.abc.OpcodeAPI`

A class representing an opcode.

classmethod **as_opcode** (*logic_fn*: `Callable[ComputationAPI, None]`, *mnemonic*: `str`, *gas_cost*: `int`) → `Type[T]`

Class factory method for turning vanilla functions into Opcode classes.

TransactionContextAPI

class `eth.abc.TransactionContextAPI` (*gas_price: int, origin: NewType.<locals>.new_type*)
 Immutable transaction context information that remains constant over the VM execution.

get_next_log_counter () → int
 Increment and return the log counter.

gas_price
 Return the gas price of the transaction context.

origin
 Return the origin of the transaction context.

MemoryAPI

class `eth.abc.MemoryAPI`
 A class representing the memory of the *VirtualMachineAPI*.

extend (*start_position: int, size: int*) → None
 Extend the memory from the given *start_position* to the provided size.

read (*start_position: int, size: int*) → memoryview
 Return a view into the memory

read_bytes (*start_position: int, size: int*) → bytes
 Read a value from memory and return a fresh bytes instance

write (*start_position: int, size: int, value: bytes*) → None
 Write *value* into memory.

StackAPI

class `eth.abc.StackAPI`
 A class representing the stack of the *VirtualMachineAPI*.

dup (*position: int*) → None
 Perform a DUP operation on the stack.

pop1_any () → Union[int, bytes]
 Pop and return an element from the stack. The type of each element will be int or bytes, depending on whether it was pushed with *push_bytes* or *push_int*.
 Raise *eth.exceptions.InsufficientStack* if the stack was empty.

pop1_bytes () → bytes
 Pop and return a bytes element from the stack.
 Raise *eth.exceptions.InsufficientStack* if the stack was empty.

pop1_int () → int
 Pop and return an integer from the stack.
 Raise *eth.exceptions.InsufficientStack* if the stack was empty.

pop_any (*num_items: int*) → Tuple[Union[int, bytes], ...]
 Pop and return a tuple of items of length *num_items* from the stack. The type of each element will be int or bytes, depending on whether it was pushed with *stack_push_bytes* or *stack_push_int*.
 Raise *eth.exceptions.InsufficientStack* if there are not enough items on the stack.

Items are ordered with the top of the stack as the first item in the tuple.

pop_bytes (*num_items: int*) → Tuple[bytes, ...]

Pop and return a tuple of bytes of length *num_items* from the stack.

Raise *eth.exceptions.InsufficientStack* if there are not enough items on the stack.

Items are ordered with the top of the stack as the first item in the tuple.

pop_ints (*num_items: int*) → Tuple[int, ...]

Pop and return a tuple of integers of length *num_items* from the stack.

Raise *eth.exceptions.InsufficientStack* if there are not enough items on the stack.

Items are ordered with the top of the stack as the first item in the tuple.

push_bytes (*value: bytes*) → None

Push a bytes item onto the stack.

push_int (*value: int*) → None

Push an integer item onto the stack.

swap (*position: int*) → None

Perform a SWAP operation on the stack.

CodeStreamAPI

class `eth.abc.CodeStreamAPI`

A class representing a stream of EVM code.

is_valid_opcode (*position: int*) → bool

Return True if a valid opcode exists at *position*.

peek () → int

Return the ordinal value of the byte at the current program counter.

read (*size: int*) → bytes

Read and return the code from the current position of the cursor up to *size*.

seek (*program_counter: int*) → ContextManager[`eth.abc.CodeStreamAPI`]

Return a `ContextManager` with the program counter set to *program_counter*.

StackManipulationAPI

class `eth.abc.StackManipulationAPI`

stack_pop1_any () → Union[int, bytes]

Pop one item from the stack and return the value either as byte or the ordinal value of a byte.

stack_pop1_bytes () → bytes

Pop one item from the stack and return the value as bytes.

stack_pop1_int () → int

Pop one item from the stack and return the ordinal value of the represented bytes.

stack_pop_any (*num_items: int*) → Tuple[Union[int, bytes], ...]

Pop the last *num_items* from the stack, returning a tuple with potentially mixed values of bytes or ordinal values of bytes.

stack_pop_bytes (*num_items: int*) → Tuple[bytes, ...]

Pop the last *num_items* from the stack, returning a tuple of bytes.

stack_pop_ints (*num_items: int*) → Tuple[int, ...]
 Pop the last *num_items* from the stack, returning a tuple of their ordinal values.

stack_push_bytes (*value: bytes*) → None
 Push *value* on the stack which must be a 32 byte string.

stack_push_int (*value: int*) → None
 Push *value* on the stack which must be a 256 bit integer.

ExecutionContextAPI

class eth.abc.ExecutionContextAPI

A class representing context information that remains constant over the execution of a block.

block_number
 Return the number of the block.

chain_id
 Return the id of the chain.

coinbase
 Return the coinbase address of the block.

difficulty
 Return the difficulty of the block.

gas_limit
 Return the gas limit of the block.

prev_hashes
 Return an iterable of block hashes that precede the block.

timestamp
 Return the timestamp of the block.

ComputationAPI

class eth.abc.ComputationAPI (*state: eth.abc.StateAPI, message: eth.abc.MessageAPI, transaction_context: eth.abc.TransactionContextAPI*)

The base class for all execution computations.

add_child_computation (*child_computation: eth.abc.ComputationAPI*) → None
 Add the given *child_computation*.

add_log_entry (*account: NewType.<locals>.new_type, topics: Tuple[int, ...], data: bytes*) → None
 Add a log entry.

apply_child_computation (*child_msg: eth.abc.MessageAPI*) → eth.abc.ComputationAPI
 Apply the vm message *child_msg* as a child computation.

classmethod apply_computation (*state: eth.abc.StateAPI, message: eth.abc.MessageAPI, transaction_context: eth.abc.TransactionContextAPI*) → eth.abc.ComputationAPI
 Perform the computation that would be triggered by the VM message.

apply_create_message () → eth.abc.ComputationAPI
 Execution of a VM message to create a new contract.

apply_message () → eth.abc.ComputationAPI
 Execution of a VM message.

consume_gas (*amount: int, reason: str*) → None
 Consume amount of gas from the remaining gas. Raise *eth.exceptions.OutOfGas* if there is not enough gas remaining.

extend_memory (*start_position: int, size: int*) → None
 Extend the size of the memory to be at minimum *start_position* + *size* bytes in length. Raise *eth.exceptions.OutOfGas* if there is not enough gas to pay for extending the memory.

generate_child_computation (*child_msg: eth.abc.MessageAPI*) → *eth.abc.ComputationAPI*
 Generate a child computation from the given *child_msg*.

get_accounts_for_deletion () → Tuple[Tuple[NewType.<locals>.new_type, NewType.<locals>.new_type], ...]
 Return a tuple of addresses that are registered for deletion.

get_gas_meter () → *eth.abc.GasMeterAPI*
 Return the *GasMeterAPI* of the computation.

get_gas_refund () → int
 Return the number of refunded gas.

get_gas_remaining () → int
 Return the number of remaining gas.

get_gas_used () → int
 Return the number of used gas.

get_log_entries () → Tuple[Tuple[bytes, Tuple[int, ...], bytes], ...]
 Return the log entries for this computation and its children.
 They are sorted in the same order they were emitted during the transaction processing, and include the sequential counter as the first element of the tuple representing every entry.

get_opcode_fn (*opcode: int*) → *eth.abc.OpcodeAPI*
 Return the function for the given *opcode*.

get_raw_log_entries () → Tuple[Tuple[int, bytes, Tuple[int, ...], bytes], ...]
 Return a tuple of raw log entries.

memory_read (*start_position: int, size: int*) → *memoryview*
 Read and return a view of *size* bytes from memory starting at *start_position*.

memory_read_bytes (*start_position: int, size: int*) → *bytes*
 Read and return *size* bytes from memory starting at *start_position*.

memory_write (*start_position: int, size: int, value: bytes*) → None
 Write *value* to memory at *start_position*. Require that *len(value) == size*.

prepare_child_message (*gas: int, to: NewType.<locals>.new_type, value: int, data: Union[bytes, memoryview], code: bytes, **kwargs*) → *eth.abc.MessageAPI*
 Helper method for creating a child computation.

raise_if_error () → None
 If there was an error during computation, raise it as an exception immediately.
 Raises *VMError* –

refund_gas (*amount: int*) → None
 Add amount of gas to the pool of gas marked to be refunded.

register_account_for_deletion (*beneficiary: NewType.<locals>.new_type*) → None
 Register the address of *beneficiary* for deletion.

return_gas (*amount: int*) → None
Return amount of gas to the available gas pool.

stack_dup (*position: int*) → None
Duplicate the stack item at `position` and pushes it onto the stack.

stack_swap (*position: int*) → None
Swap the item on the top of the stack with the item at `position`.

error
Return the `VMError` of the computation. Raise `AttributeError` if no error exists.

is_error
Return `True` if the computation resulted in an error.

is_origin_computation
Return `True` if this computation is the outermost computation at `depth == 0`.

is_success
Return `True` if the computation did not result in an error.

output
Get the return value of the computation.

precompiles
Return a dictionary where the keys are the addresses of precompiles and the values are the precompile functions.

should_burn_gas
Return `True` if the remaining gas should be burned.

should_erase_return_data
Return `True` if the return data should be zeroed out due to an error.

should_return_gas
Return `True` if the remaining gas should be returned.

AccountStorageDatabaseAPI

class `eth.abc.AccountStorageDatabaseAPI`
Storage cache and write batch for a single account. Changes are not merklized until `make_storage_root()` is called.

commit (*checkpoint: NewType.<locals>.new_type*) → None
Collapse changes into the given checkpoint.

delete () → None
Delete the entire storage at the account.

discard (*checkpoint: NewType.<locals>.new_type*) → None
Discard the given checkpoint.

get (*slot: int, from_journal: bool = True*) → int
Return the value at `slot`. Lookups take the journal into consideration unless `from_journal` is explicitly set to `False`.

get_changed_root () → `NewType.<locals>.new_type`
Return the changed root hash. Raise `ValidationError` if the root has not changed.

lock_changes () → None
Locks in changes to storage, typically just as a transaction starts.

This is used, for example, to look up the storage value from the start of the transaction, when calculating gas costs in EIP-2200: net gas metering.

make_storage_root () → None
Force calculation of the storage root for this account

persist (*db: eth.abc.DatabaseAPI*) → None
Persist all changes to the database.

record (*checkpoint: NewType.<locals>.new_type*) → None
Record changes into the given checkpoint.

set (*slot: int, value: int*) → None
Write value into slot.

has_changed_root
Return True if the storage root has changed.

AccountDatabaseAPI

```
class eth.abc.AccountDatabaseAPI (db: eth.abc.AtomicDatabaseAPI,
                                  state_root: NewType.<locals>.new_type =
                                  b'Vxe8x1fx17x1bxccUxa6xffx83Exe6x92xc0xf8n[Hxe0x1bx99lxadxc0x01b/xb5xe3cxb
```

A class representing a database for accounts.

account_exists (*address: NewType.<locals>.new_type*) → bool
Return True if an account exists at address, otherwise False.

account_has_code_or_nonce (*address: NewType.<locals>.new_type*) → bool
Return True if either code or a nonce exists at address.

account_is_empty (*address: NewType.<locals>.new_type*) → bool
Return True if an account exists at address.

commit (*checkpoint: NewType.<locals>.new_type*) → None
Collapse changes into checkpoint.

delete_account (*address: NewType.<locals>.new_type*) → None
Delete the account at address.

delete_code (*address: NewType.<locals>.new_type*) → None
Delete the code at address.

delete_storage (*address: NewType.<locals>.new_type*) → None
Delete the storage at address.

discard (*checkpoint: NewType.<locals>.new_type*) → None
Discard the given checkpoint.

get_balance (*address: NewType.<locals>.new_type*) → int
Return the balance at address.

get_code (*address: NewType.<locals>.new_type*) → bytes
Return the code at the given address.

get_code_hash (*address: NewType.<locals>.new_type*) → NewType.<locals>.new_type
Return the hash of the code at address.

get_nonce (*address: NewType.<locals>.new_type*) → int
Return the nonce for address.

get_storage (*address: NewType.<locals>.new_type, slot: int, from_journal: bool = True*) → int
Return the value stored at `slot` for the given `address`. Take the `journal` into consideration unless `from_journal` is set to `False`.

has_root (*state_root: bytes*) → bool
Return `True` if the `state_root` exists, otherwise `False`.

increment_nonce (*address: NewType.<locals>.new_type*) → None
Increment the nonce for `address`.

make_state_root () → *NewType.<locals>.new_type*
Generate the state root with all the current changes in `AccountDB`

Current changes include every pending change to storage, as well as all account changes. After generating all the required tries, the final account state root is returned.

This is an expensive operation, so should be called as little as possible. For example, pre-Byzantium, this is called after every transaction, because we need the state root in each receipt. Byzantium+, we only need state roots at the end of the block, so we *only* call it right before persistence.

Returns the new state root

persist () → None
Send changes to underlying database, including the trie state so that it will forever be possible to read the trie from this checkpoint.

make_state_root () must be explicitly called before this method. Otherwise `persist` will raise a `ValidationError`.

record () → *NewType.<locals>.new_type*
Create and return a new checkpoint.

set_balance (*address: NewType.<locals>.new_type, balance: int*) → None
Set `balance` as the new balance for `address`.

set_code (*address: NewType.<locals>.new_type, code: bytes*) → None
Set `code` as the new code at `address`.

set_nonce (*address: NewType.<locals>.new_type, nonce: int*) → None
Set `nonce` as the new nonce for `address`.

set_storage (*address: NewType.<locals>.new_type, slot: int, value: int*) → None
Write `value` into `slot` for the given `address`.

touch_account (*address: NewType.<locals>.new_type*) → None
Touch the account at `address`.

state_root
Return the state root hash.

TransactionExecutorAPI

class `eth.abc.TransactionExecutorAPI` (*vm_state: eth.abc.StateAPI*)
A class providing APIs to execute transactions on VM state.

build_computation (*message: eth.abc.MessageAPI, transaction: eth.abc.SignedTransactionAPI*) → *eth.abc.ComputationAPI*
Apply the `message` to the VM and use the given `transaction` to retrieve the context from.

build_evm_message (*transaction: eth.abc.SignedTransactionAPI*) → *eth.abc.MessageAPI*
Build and return a *MessageAPI* from the given `transaction`.

finalize_computation (*transaction*: *eth.abc.SignedTransactionAPI*, *computation*: *eth.abc.ComputationAPI*) → *eth.abc.ComputationAPI*
 Finalize the transaction.

validate_transaction (*transaction*: *eth.abc.SignedTransactionAPI*) → None
 Validate the given transaction. Raise a `ValidationError` if the transaction is invalid.

ConfigurableAPI

class `eth.abc.ConfigurableAPI`
 A class providing inline subclassing.

StateAPI

class `eth.abc.StateAPI` (*db*: *eth.abc.AtomicDatabaseAPI*, *execution_context*: *eth.abc.ExecutionContextAPI*, *state_root*: *bytes*)
 The base class that encapsulates all of the various moving parts related to the state of the VM during execution. Each *VirtualMachineAPI* must be configured with a subclass of the *StateAPI*.

Note: Each *StateAPI* class must be configured with:

- `computation_class`: The *ComputationAPI* class for vm execution.
 - `transaction_context_class`: The *TransactionContextAPI* class for vm execution.
-

account_exists (*address*: *NewType.<locals>.new_type*) → bool
 Return True if an account exists at address.

account_is_empty (*address*: *NewType.<locals>.new_type*) → bool
 Return True if the account at address is empty, otherwise False.

apply_transaction (*transaction*: *eth.abc.SignedTransactionAPI*) → *eth.abc.ComputationAPI*
 Apply transaction to the vm state

Parameters `transaction` – the transaction to apply

Returns the computation

commit (*snapshot*: *Tuple[NewType.<locals>.new_type, uuid.UUID]*) → None
 Commit the journal to the point where the snapshot was taken. This merges in any changes that were recorded since the snapshot.

costless_execute_transaction (*transaction*: *eth.abc.SignedTransactionAPI*) → *eth.abc.ComputationAPI*
 Execute the given transaction with a gas price of 0.

delete_account (*address*: *NewType.<locals>.new_type*) → None
 Delete the account at the given address.

delete_code (*address*: *NewType.<locals>.new_type*) → None
 Delete the code at address.

delete_storage (*address*: *NewType.<locals>.new_type*) → None
 Delete the storage at address

delta_balance (*address*: *NewType.<locals>.new_type*, *delta*: *int*) → None
 Apply delta to the balance at address.

classmethod `get_account_db_class` () → Type[eth.abc.AccountDatabaseAPI]
 Return the *AccountDatabaseAPI* class that the state class uses.

get_ancestor_hash (*block_number*: *NewType.<locals>.new_type*) → *NewType.<locals>.new_type*
 Return the hash for the ancestor block with number *block_number*. Return the empty bytestring `b''` if the block number is outside of the range of available block numbers (typically the last 255 blocks).

get_balance (*address*: *NewType.<locals>.new_type*) → int
 Return the balance for the account at *address*.

get_code (*address*: *NewType.<locals>.new_type*) → bytes
 Return the code at *address*.

get_code_hash (*address*: *NewType.<locals>.new_type*) → *NewType.<locals>.new_type*
 Return the hash of the code at *address*.

get_computation (*message*: *eth.abc.MessageAPI*, *transaction_context*: *eth.abc.TransactionContextAPI*) → *eth.abc.ComputationAPI*
 Return a computation instance for the given *message* and *transaction_context*

get_nonce (*address*: *NewType.<locals>.new_type*) → int
 Return the nonce at *address*.

get_storage (*address*: *NewType.<locals>.new_type*, *slot*: int, *from_journal*: bool = True) → int
 Return the storage at *slot* for *address*.

classmethod `get_transaction_context` (*transaction*: *eth.abc.SignedTransactionAPI*) → *eth.abc.TransactionContextAPI*
 Return the *TransactionContextAPI* for the given *transaction*

classmethod `get_transaction_context_class` () → Type[eth.abc.TransactionContextAPI]
 Return the *BaseTransactionContext* class that the state class uses.

get_transaction_executor () → *eth.abc.TransactionExecutorAPI*
 Return the transaction executor.

has_code_or_nonce (*address*: *NewType.<locals>.new_type*) → bool
 Return True if either a nonce or code exists at the given *address*.

increment_nonce (*address*: *NewType.<locals>.new_type*) → None
 Increment the nonce at *address*.

lock_changes () → None
 Locks in all changes to state, typically just as a transaction starts.

 This is used, for example, to look up the storage value from the start of the transaction, when calculating gas costs in EIP-2200: net gas metering.

make_state_root () → *NewType.<locals>.new_type*
 Create and return the state root.

override_transaction_context (*gas_price*: int) → ContextManager[None]
 Return a *ContextManager* that overwrites the current transaction context, applying the given *gas_price*.

persist () → None
 Persist the current state to the database.

revert (*snapshot*: Tuple[*NewType.<locals>.new_type*, *uuid.UUID*]) → None
 Revert the VM to the state at the snapshot

set_balance (*address*: *NewType.<locals>.new_type*, *balance*: int) → None
 Set *balance* to the balance at *address*.

set_code (*address: NewType.<locals>.new_type, code: bytes*) → None
 Set code as the new code at address.

set_nonce (*address: NewType.<locals>.new_type, nonce: int*) → None
 Set nonce as the new nonce at address.

set_storage (*address: NewType.<locals>.new_type, slot: int, value: int*) → None
 Write value to the given slot at address.

snapshot () → Tuple[NewType.<locals>.new_type, uuid.UUID]
 Perform a full snapshot of the current state.

Snapshots are a combination of the `state_root` at the time of the snapshot and the checkpoint from the journaled DB.

touch_account (*address: NewType.<locals>.new_type*) → None
 Touch the account at the given address.

validate_transaction (*transaction: eth.abc.SignedTransactionAPI*) → None
 Validate the given transaction.

block_number
 Return the current `block_number` from the current `execution_context`

coinbase
 Return the current `coinbase` from the current `execution_context`

difficulty
 Return the current `difficulty` from the current `execution_context`

gas_limit
 Return the current `gas_limit` from the current `transaction_context`

logger
 Return the logger.

state_root
 Return the current `state_root` from the underlying database

timestamp
 Return the current `timestamp` from the current `execution_context`

VirtualMachineAPI

```
class eth.abc.VirtualMachineAPI (header: eth.abc.BlockHeaderAPI, chaindb:
                                eth.abc.ChainDatabaseAPI, consensus_context:
                                eth.abc.ConsensusContextAPI)
```

The `VirtualMachineAPI` class represents the Chain rules for a specific protocol definition such as the Frontier or Homestead network.

Note: Each `VirtualMachineAPI` class must be configured with:

- `block_class`: The `BlockAPI` class for blocks in this VM ruleset.
 - `_state_class`: The `StateAPI` class used by this VM for execution.
-

add_receipt_to_header (*old_header: eth.abc.BlockHeaderAPI, receipt: eth.abc.ReceiptAPI*) → eth.abc.BlockHeaderAPI
 Apply the receipt to the old header, and return the resulting header. This may have storage-related side-

effects. For example, pre-Byzantium, the state root hash is included in the receipt, and so must be stored into the database.

apply_all_transactions (*transactions*: Sequence[eth.abc.SignedTransactionAPI], *base_header*: eth.abc.BlockHeaderAPI) → Tuple[eth.abc.BlockHeaderAPI, Tuple[eth.abc.ReceiptAPI, ...], Tuple[eth.abc.ComputationAPI, ...]]

Determine the results of applying all transactions to the base header. This does *not* update the current block or header of the VM.

Parameters

- **transactions** – an iterable of all transactions to apply
- **base_header** – the starting header to apply transactions to

Returns the final header, the receipts of each transaction, and the computations

apply_transaction (*header*: eth.abc.BlockHeaderAPI, *transaction*: eth.abc.SignedTransactionAPI) → Tuple[eth.abc.ReceiptAPI, eth.abc.ComputationAPI]

Apply the transaction to the current block. This is a wrapper around `apply_transaction()` with some extra orchestration logic.

Parameters

- **header** – header of the block before application
- **transaction** – to apply

classmethod build_state (*db*: eth.abc.AtomicDatabaseAPI, *header*: eth.abc.BlockHeaderAPI, *chain_context*: eth.abc.ChainContextAPI, *previous_hashes*: Iterable[NewType.<locals>.new_type] = ()) → eth.abc.StateAPI

You probably want `VM().state` instead of this.

Occasionally, you want to build custom state against a particular header and DB, even if you don't have the VM initialized. This is a convenience method to do that.

classmethod compute_difficulty (*parent_header*: eth.abc.BlockHeaderAPI, *timestamp*: int) → int

Compute the difficulty for a block header.

Parameters

- **parent_header** – the parent header
- **timestamp** – the timestamp of the child header

configure_header (***header_params*) → eth.abc.BlockHeaderAPI

Setup the current header with the provided parameters. This can be used to set fields like the gas limit or timestamp to value different than their computed defaults.

static create_execution_context (*header*: eth.abc.BlockHeaderAPI, *prev_hashes*: Iterable[NewType.<locals>.new_type], *chain_context*: eth.abc.ChainContextAPI) → eth.abc.ExecutionContextAPI

Create and return the `ExecutionContextAPI` for the given header, iterable of block hashes that precede the block and the `chain_context`.

classmethod create_header_from_parent (*parent_header*: eth.abc.BlockHeaderAPI, ***header_params*) → eth.abc.BlockHeaderAPI

Creates and initializes a new block header from the provided `parent_header`.

create_transaction (**args*, ***kwargs*) → eth.abc.SignedTransactionAPI

Proxy for instantiating a signed transaction for this VM.

classmethod create_unsigned_transaction (*, *nonce*: int, *gas_price*: int, *gas*: int, *to*: *NewType.<locals>.new_type*, *value*: int, *data*: bytes) → eth.abc.UnsignedTransactionAPI

Proxy for instantiating an unsigned transaction for this VM.

execute_bytecode (*origin*: *NewType.<locals>.new_type*, *gas_price*: int, *gas*: int, *to*: *NewType.<locals>.new_type*, *sender*: *NewType.<locals>.new_type*, *value*: int, *data*: bytes, *code*: bytes, *code_address*: *NewType.<locals>.new_type* = None) → eth.abc.ComputationAPI

Execute raw bytecode in the context of the current state of the virtual machine.

finalize_block (*block*: eth.abc.BlockAPI) → eth.abc.BlockAPI

Perform any finalization steps like awarding the block mining reward, and persisting the final state root.

classmethod generate_block_from_parent_header_and_coinbase (*parent_header*: eth.abc.BlockHeaderAPI, *coinbase*: *NewType.<locals>.new_type*) → eth.abc.BlockAPI

Generate block from parent header and coinbase.

get_block () → eth.abc.BlockAPI

Return the current block.

classmethod get_block_class () → Type[eth.abc.BlockAPI]

Return the `Block` class that this VM uses for blocks.

static get_block_reward () → int

Return the amount in **wei** that should be given to a miner as a reward for this block.

Note: This is an abstract method that must be implemented in subclasses

get_header () → eth.abc.BlockHeaderAPI

Return the current header.

classmethod get_nephew_reward () → int

Return the reward which should be given to the miner of the given *nephew*.

Note: This is an abstract method that must be implemented in subclasses

classmethod get_prev_hashes (*last_block_hash*: *NewType.<locals>.new_type*, *chaindb*: eth.abc.ChainDatabaseAPI) → Optional[Iterable[*NewType.<locals>.new_type*]]

Return an iterable of block hashes that precede the block with the given *last_block_hash*.

classmethod get_state_class () → Type[eth.abc.StateAPI]

Return the class that this VM uses for states.

classmethod get_transaction_class () → Type[eth.abc.SignedTransactionAPI]

Return the class that this VM uses for transactions.

static get_uncle_reward (*block_number*: *NewType.<locals>.new_type*, *uncle*: eth.abc.BlockAPI) → int

Return the reward which should be given to the miner of the given *uncle*.

Note: This is an abstract method that must be implemented in subclasses

import_block (*block*: *eth.abc.BlockAPI*) → *eth.abc.BlockAPI*

Import the given block to the chain.

make_receipt (*base_header*: *eth.abc.BlockHeaderAPI*, *transaction*: *eth.abc.SignedTransactionAPI*,
computation: *eth.abc.ComputationAPI*, *state*: *eth.abc.StateAPI*) →
eth.abc.ReceiptAPI

Generate the receipt resulting from applying the transaction.

Parameters

- **base_header** – the header of the block before the transaction was applied.
- **transaction** – the transaction used to generate the receipt
- **computation** – the result of running the transaction computation
- **state** – the resulting state, after executing the computation

Returns receipt

mine_block (**args*, ***kwargs*) → *eth.abc.BlockAPI*

Mine the current block. Proxies to `self.pack_block` method.

pack_block (*block*: *eth.abc.BlockAPI*, **args*, ***kwargs*) → *eth.abc.BlockAPI*

Pack block for mining.

Parameters

- **coinbase** (*bytes*) – 20-byte public address to receive block reward
- **uncles_hash** (*bytes*) – 32 bytes
- **state_root** (*bytes*) – 32 bytes
- **transaction_root** (*bytes*) – 32 bytes
- **receipt_root** (*bytes*) – 32 bytes
- **bloom** (*int*) –
- **gas_used** (*int*) –
- **extra_data** (*bytes*) – 32 bytes
- **mix_hash** (*bytes*) – 32 bytes
- **nonce** (*bytes*) – 8 bytes

set_block_transactions (*base_block*: *eth.abc.BlockAPI*, *new_header*: *eth.abc.BlockHeaderAPI*,
transactions: *Sequence[eth.abc.SignedTransactionAPI]*, *receipts*: *Se-*
quence[eth.abc.ReceiptAPI]) → *eth.abc.BlockAPI*

Create a new block with the given transactions.

state_in_temp_block () → *ContextManager[eth.abc.StateAPI]*

Return a `ContextManager` with the current state wrapped in a temporary block.

validate_block (*block*: *eth.abc.BlockAPI*) → *None*

Validate the the given block.

classmethod validate_header (*header*: *eth.abc.BlockHeaderAPI*, *parent_header*:
eth.abc.BlockHeaderAPI) → *None*

Raises `eth.exceptions.ValidationError` – if the header is not valid

classmethod validate_receipt (*receipt: eth.abc.ReceiptAPI*) → None
 Validate the given receipt.

validate_seal (*header: eth.abc.BlockHeaderAPI*) → None
 Validate the seal on the given header.

validate_seal_extension (*header: eth.abc.BlockHeaderAPI, parents: Iterable[eth.abc.BlockHeaderAPI]*) → None
 Validate the seal on the given header when all parents must be present. Parent headers that are not yet in the database must be passed as *parents*.

validate_transaction_against_header (*base_header: eth.abc.BlockHeaderAPI, transaction: eth.abc.SignedTransactionAPI*) → None
 Validate that the given transaction is valid to apply to the given header.

Parameters

- **base_header** – header before applying the transaction
- **transaction** – the transaction to validate

Raises ValidationError if the transaction is not valid to apply

classmethod validate_uncle (*block: eth.abc.BlockAPI, uncle: eth.abc.BlockHeaderAPI, uncle_parent: eth.abc.BlockHeaderAPI*) → None
 Validate the given uncle in the context of the given block.

previous_hashes

Convenience API for accessing the previous 255 block hashes.

state

Return the current state.

HeaderChainAPI

class eth.abc.HeaderChainAPI (*base_db: eth.abc.AtomicDatabaseAPI, header: eth.abc.BlockHeaderAPI = None*)

Like *eth.abc.ChainAPI* but does only support headers, not entire blocks.

classmethod from_genesis_header (*base_db: eth.abc.AtomicDatabaseAPI, genesis_header: eth.abc.BlockHeaderAPI*) → eth.abc.HeaderChainAPI
 Initialize the chain from the genesis header.

get_block_header_by_hash (*block_hash: NewType.<locals>.new_type*) → eth.abc.BlockHeaderAPI
 Direct passthrough to *headerdb*

get_canonical_block_hash (*block_number: NewType.<locals>.new_type*) → NewType.<locals>.new_type
 Direct passthrough to *headerdb*

get_canonical_block_header_by_number (*block_number: NewType.<locals>.new_type*) → eth.abc.BlockHeaderAPI
 Direct passthrough to *headerdb*

get_canonical_head () → eth.abc.BlockHeaderAPI
 Direct passthrough to *headerdb*

classmethod get_headerdb_class () → Type[eth.abc.HeaderDatabaseAPI]
 Return the class which should be used for the *headerdb*

header_exists (*block_hash: NewType.<locals>.new_type*) → bool
 Direct passthrough to *headerdb*

import_header (*header*: *eth.abc.BlockHeaderAPI*) → Tuple[Tuple[*eth.abc.BlockHeaderAPI*, ...], Tuple[*eth.abc.BlockHeaderAPI*, ...]]
 Direct passthrough to *headerdb*

Also updates the local *header* property to be the latest canonical head.

Returns an iterable of headers representing the headers that are newly part of the canonical chain.

- If the imported header is not part of the canonical chain then an empty tuple will be returned.
- If the imported header simply extends the canonical chain then a length-1 tuple with the imported header will be returned.
- If the header is part of a non-canonical chain which overtakes the current canonical chain then the returned tuple will contain the headers which are newly part of the canonical chain.

ChainAPI

class *eth.abc.ChainAPI*

A Chain is a combination of one or more VM classes. Each VM is associated with a range of blocks. The Chain class acts as a wrapper around these other VM classes, delegating operations to the appropriate VM depending on the current block number.

build_block_with_transactions (*transactions*: Tuple[*eth.abc.SignedTransactionAPI*, ...], *parent_header*: *eth.abc.BlockHeaderAPI* = None) → Tuple[*eth.abc.BlockAPI*, Tuple[*eth.abc.ReceiptAPI*, ...], Tuple[*eth.abc.ComputationAPI*, ...]]

Generate a block with the provided transactions. This does *not* import that block into your chain. If you want this new block in your chain, run *import_block()* with the result block from this method.

Parameters

- **transactions** – an iterable of transactions to insert to the block
- **parent_header** – parent of the new block – or canonical head if None

Returns (new block, receipts, computations)

create_header_from_parent (*parent_header*: *eth.abc.BlockHeaderAPI*, ***header_params*) → *eth.abc.BlockHeaderAPI*
 Passthrough helper to the VM class of the block descending from the given header.

create_transaction (**args*, ***kwargs*) → *eth.abc.SignedTransactionAPI*
 Passthrough helper to the current VM class.

create_unsigned_transaction (***, *nonce*: *int*, *gas_price*: *int*, *gas*: *int*, *to*: *NewType.<locals>.new_type*, *value*: *int*, *data*: *bytes*) → *eth.abc.UnsignedTransactionAPI*
 Passthrough helper to the current VM class.

estimate_gas (*transaction*: *eth.abc.SignedTransactionAPI*, *at_header*: *eth.abc.BlockHeaderAPI* = None) → *int*
 Return an estimation of the amount of gas the given *transaction* will use if executed on top of the block specified by *at_header*.

classmethod from_genesis (*base_db*: *eth.abc.AtomicDatabaseAPI*, *genesis_params*: Dict[*str*, Union[*int*, None, *NewType.<locals>.new_type*, *bytes*, *NewType.<locals>.new_type*, *NewType.<locals>.new_type*]], *genesis_state*: Dict[*NewType.<locals>.new_type*, *eth.typing.AccountDetails*] = None) → *eth.abc.ChainAPI*
 Initialize the Chain from a genesis state.

classmethod from_genesis_header (*base_db: eth.abc.AtomicDatabaseAPI, genesis_header: eth.abc.BlockHeaderAPI*) → eth.abc.ChainAPI
 Initialize the chain from the genesis header.

get_ancestors (*limit: int, header: eth.abc.BlockHeaderAPI*) → Tuple[eth.abc.BlockAPI, ...]
 Return *limit* number of ancestor blocks from the current canonical head.

get_block () → eth.abc.BlockAPI
 Return the current block at the tip of the chain.

get_block_by_hash (*block_hash: NewType.<locals>.new_type*) → eth.abc.BlockAPI
 Return the requested block as specified by *block_hash*.

get_block_by_header (*block_header: eth.abc.BlockHeaderAPI*) → eth.abc.BlockAPI
 Return the requested block as specified by the *block_header*.

get_block_header_by_hash (*block_hash: NewType.<locals>.new_type*) → eth.abc.BlockHeaderAPI
 Return the requested block header as specified by *block_hash*. Raise `BlockNotFound` if no block header with the given hash exists in the db.

get_canonical_block_by_number (*block_number: NewType.<locals>.new_type*) → eth.abc.BlockAPI
 Return the block with the given *block_number* in the canonical chain.
 Raise `BlockNotFound` if no block with the given *block_number* exists in the canonical chain.

get_canonical_block_hash (*block_number: NewType.<locals>.new_type*) → NewType.<locals>.new_type
 Return the block hash with the given *block_number* in the canonical chain.
 Raise `BlockNotFound` if there's no block with the given number in the canonical chain.

get_canonical_block_header_by_number (*block_number: NewType.<locals>.new_type*) → eth.abc.BlockHeaderAPI
 Return the block header with the given number in the canonical chain.
 Raise `HeaderNotFound` if there's no block header with the given number in the canonical chain.

get_canonical_head () → eth.abc.BlockHeaderAPI
 Return the block header at the canonical chain head.
 Raise `CanonicalHeadNotFound` if there's no head defined for the canonical chain.

get_canonical_transaction (*transaction_hash: NewType.<locals>.new_type*) → eth.abc.SignedTransactionAPI
 Return the requested transaction as specified by the *transaction_hash* from the canonical chain.
 Raise `TransactionNotFound` if no transaction with the specified hash is found in the canonical chain.

get_canonical_transaction_by_index (*block_number: NewType.<locals>.new_type, index: int*) → eth.abc.SignedTransactionAPI
 Return the requested transaction as specified by the *block_number* and *index* from the canonical chain.
 Raise `TransactionNotFound` if no transaction exists at *index* at *block_number* in the canonical chain.

get_canonical_transaction_index (*transaction_hash: NewType.<locals>.new_type*) → Tuple[NewType.<locals>.new_type, int]
 Return a 2-tuple of (*block_number*, *transaction_index*) indicating which block the given transaction can be found in and at what index in the block transactions.
 Raise `TransactionNotFound` if the transaction does not exist in the canonical chain.

classmethod `get_chaindb_class()` → Type[eth.abc.ChainDatabaseAPI]
 Return the class for the used *ChainDatabaseAPI*.

get_score (*block_hash: NewType.<locals>.new_type*) → int
 Return the difficulty score of the block with the given `block_hash`.
 Raise `HeaderNotFound` if there is no matching block hash.

get_transaction_receipt (*transaction_hash: NewType.<locals>.new_type*) → eth.abc.ReceiptAPI
 Return the requested receipt for the transaction as specified by the `transaction_hash`.
 Raise `ReceiptNotFound` if not receipt for the specified `transaction_hash` is found in the canonical chain.

get_transaction_receipt_by_index (*block_number: NewType.<locals>.new_type, index: int*) → eth.abc.ReceiptAPI
 Return the requested receipt for the transaction as specified by the `block_number` and `index`.
 Raise `ReceiptNotFound` if not receipt for the specified `block_number` and `index` is found in the canonical chain.

get_transaction_result (*transaction: eth.abc.SignedTransactionAPI, at_header: eth.abc.BlockHeaderAPI*) → bytes
 Return the result of running the given transaction. This is referred to as a *call()* in web3.

get_vm (*header: eth.abc.BlockHeaderAPI = None*) → eth.abc.VirtualMachineAPI
 Return the VM instance for the given header.

classmethod `get_vm_class` (*header: eth.abc.BlockHeaderAPI*) → Type[eth.abc.VirtualMachineAPI]
 Return the VM class for the given header

classmethod `get_vm_class_for_block_number` (*block_number: NewType.<locals>.new_type*) → Type[eth.abc.VirtualMachineAPI]
 Return the VM class for the given `block_number`

import_block (*block: eth.abc.BlockAPI, perform_validation: bool = True*) → eth.abc.BlockImportResult
 Import the given `block` and return a 3-tuple

- the imported block
- a tuple of blocks which are now part of the canonical chain.
- a tuple of blocks which were canonical and now are no longer canonical.

validate_block (*block: eth.abc.BlockAPI*) → None
 Validate a block that is either being mined or imported.
 Since block validation (specifically the uncle validation) must have access to the ancestor blocks, this validation must occur at the Chain level.
 Cannot be used to validate genesis block.

validate_chain (*root: eth.abc.BlockHeaderAPI, descendants: Tuple[eth.abc.BlockHeaderAPI, ...], seal_check_random_sample_rate: int = 1*) → None
 Validate that all of the descendents are valid, given that the root header is valid.
 By default, check the seal validity (Proof-of-Work on Ethereum 1.x mainnet) of all headers. This can be expensive. Instead, check a random sample of seals using `seal_check_random_sample_rate`.

validate_chain_extension (*headers: Tuple[eth.abc.BlockHeaderAPI, ...]*) → None
 Validate a chain of headers under the assumption that the entire chain of headers is present. Headers that

are not already in the database must exist in `headers`. Calling this API is not a replacement for calling `validate_chain()`, it is an additional API to call at a different stage of header processing to enable consensus schemes where the consensus can not be verified out of order.

validate_gaslimit (*header*: *eth.abc.BlockHeaderAPI*) → None
 Validate the gas limit on the given header.

validate_receipt (*receipt*: *eth.abc.ReceiptAPI*, *at_header*: *eth.abc.BlockHeaderAPI*) → None
 Validate the given receipt at the given header.

validate_seal (*header*: *eth.abc.BlockHeaderAPI*) → None
 Validate the seal on the given header.

validate_uncles (*block*: *eth.abc.BlockAPI*) → None
 Validate the uncles for the given block.

MiningChainAPI

class `eth.abc.MiningChainAPI` (*base_db*: *eth.abc.AtomicDatabaseAPI*, *header*:
eth.abc.BlockHeaderAPI = None)

Like `ChainAPI` but with APIs to create blocks incrementally.

apply_transaction (*transaction*: *eth.abc.SignedTransactionAPI*) → `Tuple[eth.abc.BlockAPI, eth.abc.ReceiptAPI, eth.abc.ComputationAPI]`
 Apply the transaction to the current tip block.

WARNING: ReceiptAPI and Transaction trie generation is computationally heavy and incurs significant performance overhead.

mine_block (**args*, ***kwargs*) → `eth.abc.BlockAPI`
 Mines the current block. Proxies to the current Virtual Machine. See VM. `mine_block()`

3.6.2 Chain

BaseChain

class `eth.chains.base.BaseChain`

The base class for all Chain objects

classmethod `get_vm_class` (*header*: *eth.abc.BlockHeaderAPI*) →
`Type[eth.abc.VirtualMachineAPI]`
 Return the VM class for the given header

classmethod `get_vm_class_for_block_number` (*block_number*: *New-Type.<locals>.new_type*) →
`Type[eth.abc.VirtualMachineAPI]`
 Return the VM class for the given `block_number`

validate_chain (*root*: *eth.abc.BlockHeaderAPI*, *descendants*: `Tuple[eth.abc.BlockHeaderAPI, ...]`,
seal_check_random_sample_rate: `int = 1`) → None
 Validate that all of the descendents are valid, given that the root header is valid.

By default, check the seal validity (Proof-of-Work on Ethereum 1.x mainnet) of all headers. This can be expensive. Instead, check a random sample of seals using `seal_check_random_sample_rate`.

validate_chain_extension (*headers*: `Tuple[eth.abc.BlockHeaderAPI, ...]`) → None
 Validate a chain of headers under the assumption that the entire chain of headers is present. Headers that are not already in the database must exist in `headers`. Calling this API is not a replacement for calling

`validate_chain()`, it is an additional API to call at a different stage of header processing to enable consensus schemes where the consensus can not be verified out of order.

Chain

class `eth.chains.base.Chain` (*base_db*: `eth.abc.AtomicDatabaseAPI`)

chaindb_class

alias of `eth.db.chain.ChainDB`

consensus_context_class

alias of `eth.consensus.context.ConsensusContext`

build_block_with_transactions (*transactions*: `Sequence[eth.abc.SignedTransactionAPI]`,
parent_header: `eth.abc.BlockHeaderAPI = None`) →
`Tuple[eth.abc.BlockAPI, Tuple[eth.abc.ReceiptAPI, ...],`
`Tuple[eth.abc.ComputationAPI, ...]]`

Generate a block with the provided transactions. This does *not* import that block into your chain. If you want this new block in your chain, run `import_block()` with the result block from this method.

Parameters

- **transactions** – an iterable of transactions to insert to the block
- **parent_header** – parent of the new block – or canonical head if `None`

Returns (new block, receipts, computations)

create_header_from_parent (*parent_header*: `eth.abc.BlockHeaderAPI`, ***header_params*) →
`eth.abc.BlockHeaderAPI`
Passthrough helper to the VM class of the block descending from the given header.

create_transaction (**args*, ***kwargs*) → `eth.abc.SignedTransactionAPI`
Passthrough helper to the current VM class.

create_unsigned_transaction (***, *nonce*: `int`, *gas_price*: `int`, *gas*: `int`, *to*: `NewType.<locals>.new_type`,
value: `int`, *data*: `bytes`) →
`eth.abc.UnsignedTransactionAPI`
Passthrough helper to the current VM class.

ensure_header (*header*: `eth.abc.BlockHeaderAPI = None`) → `eth.abc.BlockHeaderAPI`
Return header if it is not `None`, otherwise return the header of the canonical head.

estimate_gas (*transaction*: `eth.abc.SignedTransactionAPI`, *at_header*: `eth.abc.BlockHeaderAPI = None`) → `int`
Return an estimation of the amount of gas the given `transaction` will use if executed on top of the block specified by `at_header`.

classmethod from_genesis (*base_db*: `eth.abc.AtomicDatabaseAPI`, *genesis_params*: `Dict[str, Union[int, None, NewType.<locals>.new_type, bytes, NewType.<locals>.new_type, NewType.<locals>.new_type]]`,
genesis_state: `Dict[NewType.<locals>.new_type, eth.typing.AccountDetails] = None`) → `eth.chains.base.BaseChain`
Initialize the Chain from a genesis state.

classmethod from_genesis_header (*base_db*: `eth.abc.AtomicDatabaseAPI`, *genesis_header*: `eth.abc.BlockHeaderAPI`) → `eth.chains.base.BaseChain`
Initialize the chain from the genesis header.

get_ancestors (*limit*: `int`, *header*: `eth.abc.BlockHeaderAPI`) → `Tuple[eth.abc.BlockAPI, ...]`
Return *limit* number of ancestor blocks from the current canonical head.

get_block () → eth.abc.BlockAPI
 Return the current block at the tip of the chain.

get_block_by_hash (*block_hash: NewType.<locals>.new_type*) → eth.abc.BlockAPI
 Return the requested block as specified by `block_hash`.

get_block_by_header (*block_header: eth.abc.BlockHeaderAPI*) → eth.abc.BlockAPI
 Return the requested block as specified by the `block_header`.

get_block_header_by_hash (*block_hash: NewType.<locals>.new_type*) → eth.abc.BlockHeaderAPI
 Return the requested block header as specified by `block_hash`. Raise `BlockNotFound` if no block header with the given hash exists in the db.

get_canonical_block_by_number (*block_number: NewType.<locals>.new_type*) → eth.abc.BlockAPI
 Return the block with the given `block_number` in the canonical chain.
 Raise `BlockNotFound` if no block with the given `block_number` exists in the canonical chain.

get_canonical_block_hash (*block_number: NewType.<locals>.new_type*) → NewType.<locals>.new_type
 Return the block hash with the given `block_number` in the canonical chain.
 Raise `BlockNotFound` if there's no block with the given number in the canonical chain.

get_canonical_block_header_by_number (*block_number: NewType.<locals>.new_type*) → eth.abc.BlockHeaderAPI
 Return the block header with the given number in the canonical chain.
 Raise `HeaderNotFound` if there's no block header with the given number in the canonical chain.

get_canonical_head () → eth.abc.BlockHeaderAPI
 Return the block header at the canonical chain head.
 Raise `CanonicalHeadNotFound` if there's no head defined for the canonical chain.

get_canonical_transaction (*transaction_hash: NewType.<locals>.new_type*) → eth.abc.SignedTransactionAPI
 Return the requested transaction as specified by the `transaction_hash` from the canonical chain.
 Raise `TransactionNotFound` if no transaction with the specified hash is found in the canonical chain.

get_canonical_transaction_by_index (*block_number: NewType.<locals>.new_type, index: int*) → eth.abc.SignedTransactionAPI
 Return the requested transaction as specified by the `block_number` and `index` from the canonical chain.
 Raise `TransactionNotFound` if no transaction exists at `index` at `block_number` in the canonical chain.

get_canonical_transaction_index (*transaction_hash: NewType.<locals>.new_type*) → Tuple[NewType.<locals>.new_type, int]
 Return a 2-tuple of (`block_number`, `transaction_index`) indicating which block the given transaction can be found in and at what index in the block transactions.
 Raise `TransactionNotFound` if the transaction does not exist in the canonical chain.

classmethod get_chaindb_class () → Type[eth.abc.ChainDatabaseAPI]
 Return the class for the used `ChainDatabaseAPI`.

get_score (*block_hash: NewType.<locals>.new_type*) → int
 Return the difficulty score of the block with the given `block_hash`.
 Raise `HeaderNotFound` if there is no matching block hash.

get_transaction_receipt (*transaction_hash*: *NewType.<locals>.new_type*) → *eth.abc.ReceiptAPI*
 Return the requested receipt for the transaction as specified by the `transaction_hash`.
 Raise `ReceiptNotFound` if not receipt for the specified `transaction_hash` is found in the canonical chain.

get_transaction_receipt_by_index (*block_number*: *NewType.<locals>.new_type*, *index*: *int*) → *eth.abc.ReceiptAPI*
 Return the requested receipt for the transaction as specified by the `block_number` and `index`.
 Raise `ReceiptNotFound` if not receipt for the specified `block_number` and `index` is found in the canonical chain.

get_transaction_result (*transaction*: *eth.abc.SignedTransactionAPI*, *at_header*: *eth.abc.BlockHeaderAPI*) → *bytes*
 Return the result of running the given transaction. This is referred to as a `call()` in web3.

get_vm (*at_header*: *eth.abc.BlockHeaderAPI = None*) → *eth.abc.VirtualMachineAPI*
 Return the VM instance for the given header.

import_block (*block*: *eth.abc.BlockAPI*, *perform_validation*: *bool = True*) → *eth.abc.BlockImportResult*
 Import the given `block` and return a 3-tuple

- the imported block
- a tuple of blocks which are now part of the canonical chain.
- a tuple of blocks which were canonical and now are no longer canonical.

validate_block (*block*: *eth.abc.BlockAPI*) → *None*
 Validate a block that is either being mined or imported.
 Since block validation (specifically the uncle validation) must have access to the ancestor blocks, this validation must occur at the Chain level.
 Cannot be used to validate genesis block.

validate_gaslimit (*header*: *eth.abc.BlockHeaderAPI*) → *None*
 Validate the gas limit on the given header.

validate_receipt (*receipt*: *eth.abc.ReceiptAPI*, *at_header*: *eth.abc.BlockHeaderAPI*) → *None*
 Validate the given `receipt` at the given header.

validate_seal (*header*: *eth.abc.BlockHeaderAPI*) → *None*
 Validate the seal on the given header.

validate_uncles (*block*: *eth.abc.BlockAPI*) → *None*
 Validate the uncles for the given `block`.

MiningChain

class `eth.chains.base.MiningChain` (*base_db*: *eth.abc.AtomicDatabaseAPI*, *header*: *eth.abc.BlockHeaderAPI = None*)

apply_transaction (*transaction*: *eth.abc.SignedTransactionAPI*) → *Tuple[eth.abc.BlockAPI, eth.abc.ReceiptAPI, eth.abc.ComputationAPI]*
 Apply the transaction to the current tip block.
 WARNING: ReceiptAPI and Transaction trie generation is computationally heavy and incurs significant performance overhead.

get_vm (*at_header: eth.abc.BlockHeaderAPI = None*) → eth.abc.VirtualMachineAPI
 Return the VM instance for the given header.

import_block (*block: eth.abc.BlockAPI, perform_validation: bool = True*) → eth.abc.BlockImportResult
 Import the given `block` and return a 3-tuple

- the imported block
- a tuple of blocks which are now part of the canonical chain.
- a tuple of blocks which were canonical and now are no longer canonical.

mine_block (**args, **kwargs*) → eth.abc.BlockAPI
 Mines the current block. Proxies to the current Virtual Machine. See VM. `mine_block()`

3.6.3 DataBase

Account

AccountDB

```
class eth.db.account.AccountDB (db: eth.abc.AtomicDatabaseAPI,  

                                state_root: NewType.<locals>.new_type =  

                                b'Vxe8x1fx17x1bxccUxa6xffx83Exe6x92xc0xf8n[Hxe0x1bx99lxadxc0x01b/xb5xe3cxb4!'
```

account_exists (*address: NewType.<locals>.new_type*) → bool
 Return True if an account exists at address, otherwise False.

account_has_code_or_nonce (*address: NewType.<locals>.new_type*) → bool
 Return True if either code or a nonce exists at address.

account_is_empty (*address: NewType.<locals>.new_type*) → bool
 Return True if an account exists at address.

commit (*checkpoint: NewType.<locals>.new_type*) → None
 Collapse changes into checkpoint.

delete_account (*address: NewType.<locals>.new_type*) → None
 Delete the account at address.

delete_code (*address: NewType.<locals>.new_type*) → None
 Delete the code at address.

delete_storage (*address: NewType.<locals>.new_type*) → None
 Delete the storage at address.

discard (*checkpoint: NewType.<locals>.new_type*) → None
 Discard the given checkpoint.

get_balance (*address: NewType.<locals>.new_type*) → int
 Return the balance at address.

get_code (*address: NewType.<locals>.new_type*) → bytes
 Return the code at the given address.

get_code_hash (*address: NewType.<locals>.new_type*) → NewType.<locals>.new_type
 Return the hash of the code at address.

get_nonce (*address: NewType.<locals>.new_type*) → int
 Return the nonce for address.

get_storage (*address: NewType.<locals>.new_type, slot: int, from_journal: bool = True*) → int
 Return the value stored at `slot` for the given `address`. Take the journal into consideration unless `from_journal` is set to `False`.

has_root (*state_root: bytes*) → bool
 Return `True` if the `state_root` exists, otherwise `False`.

increment_nonce (*address: NewType.<locals>.new_type*) → None
 Increment the nonce for `address`.

make_state_root () → *NewType.<locals>.new_type*
 Generate the state root with all the current changes in AccountDB

Current changes include every pending change to storage, as well as all account changes. After generating all the required tries, the final account state root is returned.

This is an expensive operation, so should be called as little as possible. For example, pre-Byzantium, this is called after every transaction, because we need the state root in each receipt. Byzantium+, we only need state roots at the end of the block, so we *only* call it right before persistence.

Returns the new state root

persist () → None
 Send changes to underlying database, including the trie state so that it will forever be possible to read the trie from this checkpoint.

make_state_root () must be explicitly called before this method. Otherwise `persist` will raise a `ValidationError`.

record () → *NewType.<locals>.new_type*
 Create and return a new checkpoint.

set_balance (*address: NewType.<locals>.new_type, balance: int*) → None
 Set `balance` as the new balance for `address`.

set_code (*address: NewType.<locals>.new_type, code: bytes*) → None
 Set `code` as the new code at `address`.

set_nonce (*address: NewType.<locals>.new_type, nonce: int*) → None
 Set `nonce` as the new nonce for `address`.

set_storage (*address: NewType.<locals>.new_type, slot: int, value: int*) → None
 Write `value` into `slot` for the given `address`.

touch_account (*address: NewType.<locals>.new_type*) → None
 Touch the account at `address`.

state_root
 Return the state root hash.

Atomic

AtomicDB

class `eth.db.atomic.AtomicDB` (*wrapped_db: eth.abc.DatabaseAPI = None*)

atomic_batch () → `Iterator[AtomicDBWriteBatch]`
 Return a `ContextManager` to write an atomic batch to the database.

class `eth.db.atomic.AtomicDBWriteBatch` (*write_target_db: eth.abc.DatabaseAPI*)

This is returned by a `BaseAtomicDB` during an `atomic_batch`, to provide a temporary view of the database, before commit.

Backends

BaseDB

class `eth.db.backends.base.BaseDB`

This is an abstract key/value lookup with all `bytes` values, with some convenience methods for databases. As much as possible, you can use a DB as if it were a `dict`.

Notable exceptions are that you cannot iterate through all values or get the length. (Unless a subclass explicitly enables it).

All subclasses must implement these methods: `__init__`, `__getitem__`, `__setitem__`, `__delitem__`

Subclasses may optionally implement an `_exists` method that is type-checked for key and value.

delete (*key: bytes*) → None

Delete the given `key` from the database.

exists (*key: bytes*) → bool

Return `True` if the `key` exists in the database, otherwise `False`.

set (*key: bytes, value: bytes*) → None

Assign the `value` to the `key`.

LevelDB

class `eth.db.backends.level.LevelDB` (*db_path: pathlib.Path = None, max_open_files: int = None*)

atomic_batch () → Generator[[`LevelDBWriteBatch`, None], None]

Return a `ContextManager` to write an atomic batch to the database.

MemoryDB

class `eth.db.backends.memory.MemoryDB` (*kv_store: Dict[bytes, bytes] = None*)

Batch

BatchDB

class `eth.db.batch.BatchDB` (*wrapped_db: eth.abc.DatabaseAPI, read_through_deletes: bool = False*)

A wrapper of basic DB objects with uncommitted DB changes stored in local cache, which represents as a dictionary of database keys and values. This class should be usable as a context manager, the changes either all fail or all succeed. Upon exiting the context, it writes all of the key value pairs from the cache into the underlying database. If any error occurred before committing phase, we would not apply commits at all.

clear () → None. Remove all items from D.

Cache

CacheDB

class `eth.db.cache.CacheDB` (*db: eth.abc.DatabaseAPI, cache_size: int = 2048*)
 Set and get decoded RLP objects, where the underlying db stores encoded objects.

Chain

ChainDB

class `eth.db.chain.ChainDB` (*db: eth.abc.AtomicDatabaseAPI*)

add_receipt (*block_header: eth.abc.BlockHeaderAPI, index_key: int, receipt: eth.abc.ReceiptAPI*)
 → `NewType.<locals>.new_type`
 Add the given receipt to the provided block header.

Return the updated `receipts_root` for updated block header.

add_transaction (*block_header: eth.abc.BlockHeaderAPI, index_key: int, transaction: eth.abc.SignedTransactionAPI*) → `NewType.<locals>.new_type`
 Add the given transaction to the provided block header.

Return the updated `transactions_root` for updated block header.

exists (*key: bytes*) → `bool`
 Return `True` if the given key exists in the database.

get (*key: bytes*) → `bytes`
 Return the value for the given key or a `KeyError` if it doesn't exist in the database.

get_block_transaction_hashes (*block_header: eth.abc.BlockHeaderAPI*) → `Tuple[NewType.<locals>.new_type, ...]`
 Returns an iterable of the transaction hashes from the block specified by the given block header.

get_block_transactions (*header: eth.abc.BlockHeaderAPI, transaction_class: Type[eth.abc.SignedTransactionAPI]*) → `Tuple[eth.abc.SignedTransactionAPI, ...]`
 Return an iterable of transactions for the block specified by the given block header.

get_block_uncles (*uncles_hash: NewType.<locals>.new_type*) → `Tuple[eth.abc.BlockHeaderAPI, ...]`
 Return an iterable of uncle headers specified by the given `uncles_hash`

get_receipt_by_index (*block_number: NewType.<locals>.new_type, receipt_index: int*) → `eth.abc.ReceiptAPI`
 Return the receipt of the transaction at specified index for the block header obtained by the specified block number

get_receipts (*header: eth.abc.BlockHeaderAPI, receipt_class: Type[eth.abc.ReceiptAPI]*) → `Iterable[eth.abc.ReceiptAPI]`
 Return a tuple of receipts for the block specified by the given block header.

get_transaction_by_index (*block_number: NewType.<locals>.new_type, transaction_index: int, transaction_class: Type[eth.abc.SignedTransactionAPI]*) → `eth.abc.SignedTransactionAPI`
 Return the transaction at the specified `transaction_index` from the block specified by `block_number` from the canonical chain.

`pending_keys ()` → Iterable[bytes]

List all the keys who have had values change. This IGNORES any keys that have been deleted.

DBDiffTracker

class `eth.db.diff.DBDiffTracker`

Records changes to a *DatabaseAPI*

If no value is available for a key, it could be for one of two reasons: - the key was never updated during tracking
- the key was deleted at some point

When getting a value, a special subtype of `KeyError` is raised on failure. The exception, *DiffMissingError*, can be used to check if the value was deleted, or never present, using `DiffMissingError.is_deleted()`.

When it's time to take the tracked changes and write them to your database, get the *DBDiff* with `DBDiffTracker.diff()` and use the attached methods.

DiffMissingError

class `eth.db.diff.DiffMissingError` (*missing_key: bytes, reason: eth.db.diff.MissingReason*)

Raised when trying to access a missing key/value pair in a *DBDiff* or *DBDiffTracker*.

Use `is_deleted` to check if the value is missing because it was deleted, or simply because it was never updated.

Header

HeaderDB

class `eth.db.header.HeaderDB` (*db: eth.abc.AtomicDatabaseAPI*)

get_block_header_by_hash (*block_hash: NewType.<locals>.new_type*) → `eth.abc.BlockHeaderAPI`
Return the block header for the given `block_hash`. Raise `HeaderNotFound` if no header with the given `block_hash` exists in the database.

get_canonical_block_hash (*block_number: NewType.<locals>.new_type*) → `NewType.<locals>.new_type`
Return the block hash for the canonical block at the given number.

Raise `BlockNotFound` if there's no block header with the given number in the canonical chain.

get_canonical_block_header_by_number (*block_number: NewType.<locals>.new_type*) → `eth.abc.BlockHeaderAPI`
Return the block header with the given number in the canonical chain.

Raise `HeaderNotFound` if there's no block header with the given number in the canonical chain.

get_canonical_head () → `eth.abc.BlockHeaderAPI`
Return the current block header at the head of the chain.

get_score (*block_hash: NewType.<locals>.new_type*) → int
Return the score for the given `block_hash`.

header_exists (*block_hash: NewType.<locals>.new_type*) → bool
Return True if the `block_hash` exists in the database, otherwise False.

persist_checkpoint_header (*header: eth.abc.BlockHeaderAPI, score: int*) → None

Persist a checkpoint header with a trusted score. Persisting the checkpoint header automatically sets it as the new canonical head.

persist_header (*header: eth.abc.BlockHeaderAPI*) → Tuple[Tuple[eth.abc.BlockHeaderAPI, ...], Tuple[eth.abc.BlockHeaderAPI, ...]]

Persist the *header* in the database. Return two iterable of headers, the first containing the new canonical header, the second containing the old canonical headers

persist_header_chain (*headers: Iterable[eth.abc.BlockHeaderAPI], genesis_parent_hash: NewType.<locals>.new_type*) → Tuple[Tuple[eth.abc.BlockHeaderAPI, ...], Tuple[eth.abc.BlockHeaderAPI, ...]]

Persist a chain of headers in the database. Return two iterable of headers, the first containing the new canonical headers, the second containing the old canonical headers

Parameters *genesis_parent_hash* – optional parent hash of the block that is treated as genesis. Providing a *genesis_parent_hash* allows storage of headers that aren't (yet) connected back to the true genesis header.

Journal

JournalDB

class eth.db.journal.**JournalDB** (*wrapped_db: eth.abc.DatabaseAPI*)

A wrapper around the basic DB objects that keeps a journal of all changes. Checkpoints can be recorded at any time. You can then commit or roll back to those checkpoints.

Discarding a checkpoint throws away all changes that happened since that checkpoint. Committing a checkpoint simply removes the option of reverting back to it later.

Nothing is written to the underlying db until *persist()* is called.

The added memory footprint for a JournalDB is one key/value stored per database key which is changed, at each checkpoint. Subsequent changes to the same key between two checkpoints will not increase the journal size, since we do not permit reverting to a place that has no checkpoint.

clear () → None

Remove all keys. Immediately after a clear, *all* getitem requests will return a `KeyError`. That includes the changes pending persist and any data in the underlying database.

(This action is journaled, like all other actions)

clear will *not* persist the emptying of all keys in the underlying DB. It only prevents any updates (or deletes!) before it from being persisted.

Any caller that wants to use *clear* must also make sure that the underlying database reflects their desired end state (maybe emptied, maybe not).

diff () → eth.db.diff.DBDiff

Generate a DBDiff of all pending changes. These are the changes that would occur if *persist()* were called.

discard (*checkpoint: NewType.<locals>.new_type*) → None

Throws away all journaled data starting at the given checkpoint

flatten () → None

Commit everything possible without persisting

persist () → None

Persist all changes in underlying db. After all changes have been written the JournalDB starts a new recording.

reset () → None

Reset the entire journal.

Schema

SchemaV1

class eth.db.schema.SchemaV1

static **make_block_hash_to_score_lookup_key** (*block_hash*: *NewType.<locals>.new_type*) → bytes

Return the lookup key to retrieve the score from a block hash.

static **make_block_number_to_hash_lookup_key** (*block_number*: *NewType.<locals>.new_type*) → bytes

Return the lookup key to retrieve a block hash from a block number.

static **make_canonical_head_hash_lookup_key** () → bytes

Return the lookup key to retrieve the canonical head from the database.

static **make_transaction_hash_to_block_lookup_key** (*transaction_hash*: *NewType.<locals>.new_type*) → bytes

Return the lookup key to retrieve a transaction key from a transaction hash.

Storage

AccountStorageDB

class eth.db.storage.AccountStorageDB (*db*: *eth.abc.AtomicDatabaseAPI*, *storage_root*: *NewType.<locals>.new_type*, *address*: *NewType.<locals>.new_type*)

commit (*checkpoint*: *NewType.<locals>.new_type*) → None

Collapse changes into the given checkpoint.

delete () → None

Delete the entire storage at the account.

discard (*checkpoint*: *NewType.<locals>.new_type*) → None

Discard the given checkpoint.

get (*slot*: *int*, *from_journal*: *bool = True*) → int

Return the value at slot. Lookups take the journal into consideration unless *from_journal* is explicitly set to *False*.

get_changed_root () → *NewType.<locals>.new_type*

Return the changed root hash. Raise *ValidationError* if the root has not changed.

lock_changes () → None

Locks in changes to storage, typically just as a transaction starts.

This is used, for example, to look up the storage value from the start of the transaction, when calculating gas costs in EIP-2200: net gas metering.

make_storage_root () → None
Force calculation of the storage root for this account

persist (*db: eth.abc.DatabaseAPI*) → None
Persist all changes to the database.

record (*checkpoint: NewType.<locals>.new_type*) → None
Record changes into the given checkpoint.

set (*slot: int, value: int*) → None
Write value into slot.

has_changed_root
Return True if the storage root has changed.

StorageLookup

```
class eth.db.storage.StorageLookup (db: eth.abc.DatabaseAPI, storage_root: New-
                                     Type.<locals>.new_type, address: New-
                                     Type.<locals>.new_type)
```

This lookup converts lookups of storage slot integers into the appropriate trie lookup. Similarly, it persists changes to the appropriate trie at write time.

StorageLookup also tracks the state roots changed since the last persist.

commit_to (*db: eth.abc.DatabaseAPI*) → None
Trying to commit changes when nothing has been written will raise a `ValidationError`

new_trie () → int
Switch to an empty trie. Save the old trie, and pending writes, in case of a revert.
Returns index for reviving the previous trie

rollback_trie (*trie_index: int*) → None
Revert back to the previous trie, using the index returned by a `new_trie()` call. The index returned by that call returns you to the trie in place *before* the call.
Parameters `trie_index` – index for reviving the previous trie

3.6.4 Exceptions

exception `eth.exceptions.BlockNotFound`
Raised when the block with the given number/hash does not exist.

exception `eth.exceptions.CanonicalHeadNotFound`
Raised when the chain has no canonical head.

exception `eth.exceptions.ContractCreationCollision`
Raised when there was an address collision during contract creation.

exception `eth.exceptions.FullStack`
Raised when the stack is full.

exception `eth.exceptions.Halt`
Raised when an opcode function halts vm execution.

- exception** `eth.exceptions.HeaderNotFound`
Raised when a header with the given number/hash does not exist.
- exception** `eth.exceptions.IncorrectContractCreationAddress`
Raised when the address provided by transaction does not match the calculated contract creation address.
- exception** `eth.exceptions.InsufficientFunds`
Raised when an account has insufficient funds to transfer the requested value.
- exception** `eth.exceptions.InsufficientStack`
Raised when the stack is empty.
- exception** `eth.exceptions.InvalidInstruction`
Raised when an opcode is invalid.
- exception** `eth.exceptions.InvalidJumpDestination`
Raised when the jump destination for a JUMPDEST operation is invalid.
- exception** `eth.exceptions.OutOfBoundsRead`
Raised when an attempt was made to read data beyond the boundaries of the buffer (such as with RETURN-DATACOPY)
- exception** `eth.exceptions.OutOfGas`
Raised when a VM execution has run out of gas.
- exception** `eth.exceptions.ParentNotFound`
Raised when the parent of a given block does not exist.
- exception** `eth.exceptions.PyEVMError`
Base class for all py-vm errors.
- exception** `eth.exceptions.ReceiptNotFound`
Raised when the Receipt with the given receipt index does not exist.
- exception** `eth.exceptions.Revert`
Raised when the REVERT opcode occurred
- exception** `eth.exceptions.StackDepthLimit`
Raised when the call stack has exceeded its maximum allowed depth.
- exception** `eth.exceptions.StateRootNotFound`
Raised when the requested state root is not present in our DB.
- exception** `eth.exceptions.TransactionNotFound`
Raised when the transaction with the given hash or block index does not exist.
- exception** `eth.exceptions.VMError`
Base class for errors raised during VM execution.
- exception** `eth.exceptions.VMNotFound`
Raised when no VM is available for the provided block number.
- exception** `eth.exceptions.WriteProtection`
Raised when an attempt to modify the state database is made while operating inside of a STATICCALL context.

3.6.5 RLP

Accounts

Logs

Log

```
class eth.rlp.logs.Log (address: bytes, topics: Tuple[int, ...], data: bytes)
```

Receipts

Receipt

```
class eth.rlp.receipts.Receipt (state_root: bytes, gas_used: int, logs: Iterable[eth.rlp.logs.Log], bloom: int = None)
```

Transactions

BaseTransactionMethods

```
class eth.rlp.transactions.BaseTransactionMethods
```

```
gas_used_by (computation: eth.abc.ComputationAPI) → int
```

Return the gas used by the given computation. In Frontier, for example, this is sum of the intrinsic cost and the gas used during computation.

```
validate () → None
```

Hook called during instantiation to ensure that all transaction parameters pass validation rules.

```
intrinsic_gas
```

Convenience property for the return value of *get_intrinsic_gas*

BaseTransactionFields

```
class eth.rlp.transactions.BaseTransactionFields (*args, **kwargs)
```

BaseTransaction

```
class eth.rlp.transactions.BaseTransaction (*args, **kwargs)
```

```
classmethod from_base_transaction (transaction: eth.abc.SignedTransactionAPI) → eth.abc.SignedTransactionAPI
```

Create a signed transaction from a base transaction.

```
validate () → None
```

Hook called during instantiation to ensure that all transaction parameters pass validation rules.

```
is_signature_valid
```

Return True if the signature is valid, otherwise False.

BaseUnsignedTransaction

```
class eth.rlp.transactions.BaseUnsignedTransaction (*args, **kwargs)
```

3.6.6 Tools

Builders

Chain Builder

The chain builder utils are intended to reduce common boilerplate for both construction of chain classes as well as building up some desired chain state.

Note: These tools are best used in conjunction with `cytoolz.pipe`.

Constructing Chain Classes

The following utilities are provided to assist with constructing a chain class.

`eth.tools.builder.chain.fork_at()`
Adds the `vm_class` to the chain's `vm_configuration`.

```
from eth.chains.base import MiningChain
from eth.tools.builder.chain import build, fork_at

FrontierOnlyChain = build(MiningChain, fork_at(FrontierVM, 0))

# these two classes are functionally equivalent.
class FrontierOnlyChain(MiningChain):
    vm_configuration = (
        (0, FrontierVM),
    )
```

Note: This function is curriable.

The following pre-curved versions of this function are available as well, one for each mainnet fork.

- `frontier_at()`
- `homestead_at()`
- `tangerine_whistle_at()`
- `spurious_dragon_at()`
- `byzantium_at()`
- `constantinople_at()`
- `petersburg_at()`
- `istanbul_at()`
- `latest_mainnet_at()` - whatever the latest mainnet VM is

`eth.tools.builder.chain.dao_fork_at()`
Set the block number on which the DAO fork will happen. Requires that a version of the *HomesteadVM* is present in the chain's `vm_configuration`

`eth.tools.builder.chain.disable_dao_fork()`
 Set the `support_dao_fork` flag to `False` on the `HomesteadVM`. Requires that presence of the `HomesteadVM` in the `vm_configuration`

`eth.tools.builder.chain.enable_pow_mining()`
 Inject on demand generation of the proof of work mining seal on newly mined blocks into each of the chain's vms.

`eth.tools.builder.chain.disable_pow_check()`
 Disable the proof of work validation check for each of the chain's vms. This allows for block mining without generation of the proof of work seal.

Note: blocks mined this way will not be importable on any chain that does not have proof of work disabled.

`eth.tools.builder.chain.name()`
 Assign the given name to the chain class.

`eth.tools.builder.chain.chain_id()`
 Set the `chain_id` for the chain class.

Initializing Chains

The following utilities are provided to assist with initializing a chain into the genesis state.

`eth.tools.builder.chain.genesis()`
 Initialize the given chain class with the given genesis header parameters and chain state.

Building Chains

The following utilities are provided to assist with building out chains of blocks.

`eth.tools.builder.chain.copy()`
 Make a copy of the chain at the given state. Actions performed on the resulting chain will not affect the original chain.

`eth.tools.builder.chain.import_block()`
 Import the provided `block` into the chain.

`eth.tools.builder.chain.import_blocks(*blocks)` → `Callable[eth.abc.ChainAPI, eth.abc.ChainAPI]`
 Variadic argument version of `import_block()`

`eth.tools.builder.chain.mine_block()`
 Mine a new block on the chain. Header parameters for the new block can be overridden using keyword arguments.

`eth.tools.builder.chain.mine_blocks()`
 Variadic argument version of `mine_block()`

`eth.tools.builder.chain.chain_split(*splits)` → `Callable[eth.abc.ChainAPI, Iterable[eth.abc.ChainAPI]]`
 Construct and execute multiple concurrent forks of the chain.

Any number of forks may be executed. For each fork, provide an iterable of commands.

Returns the resulting chain objects for each fork.

```
chain_a, chain_b = build(
    mining_chain,
    chain_split(
        (mine_block(extra_data=b'chain-a'), mine_block()),
        (mine_block(extra_data=b'chain-b'), mine_block(), mine_block()),
    ),
)
```

`eth.tools.builder.chain.at_block_number()`

Rewind the chain back to the given block number. Calls to things like `get_canonical_head` will still return the canonical head of the chain, however, you can use `mine_block` to mine fork chains.

Builder Tools

The JSON test fillers found in `eth.tools.fixtures` is a set of tools which facilitate creating standard JSON consensus tests as found in the [ethereum/tests repository](#).

Note: Only VM and state tests are supported right now.

State Test Fillers

Tests are generated in two steps.

- First, a *test filler* is written that contains a high level description of the test case.
- Subsequently, the filler is compiled to the actual test in a process called filling, mainly consisting of calculating the resulting state root.

The test builder represents each stage as a nested dictionary. Helper functions are provided to assemble the filler file step by step in the correct format. The `fill_test()` function handles compilation and takes additional parameters that can't be inferred from the filler.

Creating a Filler

Fillers are generated in a functional fashion by piping a dictionary through a sequence of functions.

```
filler = pipe(
    setup_main_filler("test"),
    pre_state(
        (sender, "balance", 1),
        (receiver, "balance", 0),
    ),
    expect(
        networks=["Frontier"],
        transaction={
            "to": receiver,
            "value": 1,
            "secretKey": sender_key,
        },
        post_state=[
            [sender, "balance", 0],
            [receiver, "balance", 1],
        ],
    ),
)
```

(continues on next page)

(continued from previous page)

```

    ]
  )
)

```

Note: Note that `setup_filler()` returns a dictionary, whereas all of the following functions such as `pre_state()`, `expect()`, `expect` to be passed a dictionary as their single argument and return an updated version of the dictionary.

`eth.tools.fixtures.fillers.common.setup_main_filler` (*name*: *str*, *environment*: *Dict[Any, Any] = None*) → *Dict[str, Dict[str, Any]]*

Kick off the filler generation process by creating the general filler scaffold with a test name and general information about the testing environment.

For tests for the main chain, the *environment* parameter is expected to be a dictionary with some or all of the following keys:

key	description
"currentCoinbase"	the coinbase address
"currentNumber"	the block number
"previousHash"	the hash of the parent block
"currentDifficulty"	the block's difficulty
"currentGasLimit"	the block's gas limit
"currentTimestamp"	the timestamp of the block

`eth.tools.fixtures.fillers.pre_state` (**raw_state*, *filler*: *Dict[str, Any]*) → *None*

Specify the state prior to the test execution. Multiple invocations don't override the state but extend it instead.

In general, the elements of *state_definitions* are nested dictionaries of the following form:

```

{
  address: {
    "nonce": <account nonce>,
    "balance": <account balance>,
    "code": <account code>,
    "storage": {
      <storage slot>: <storage value>
    }
  }
}

```

To avoid unnecessary nesting especially if only few fields per account are specified, the following and similar formats are possible as well:

```

(address, "balance", <account balance>)
(address, "storage", <storage slot>, <storage value>)
(address, "storage", {<storage slot>: <storage value>})
(address, {"balance", <account balance>})

```

`eth.tools.fixtures.fillers.execution` ()

For VM tests, specify the code that is being run as well as the current state of the EVM. State tests don't support this object. The parameter is a dictionary specifying some or all of the following keys:

key	description
"address"	the address of the account executing the code
"caller"	the caller address
"origin"	the origin address (defaulting to the caller address)
"value"	the value of the call
"data"	the data passed with the call
"gasPrice"	the gas price of the call
"gas"	the amount of gas allocated for the call
"code"	the bytecode to execute
"vyperLLLCode"	the code in Vyper LLL (compiled to bytecode automatically)

`eth.tools.fixtures.fillers.expect` (*post_state*: Dict[str, Any] = None, *networks*: Any = None, *transaction*: eth.typing.TransactionDict = None) → Callable[..., Dict[str, Any]]

Specify the expected result for the test.

For state tests, multiple expectations can be given, differing in the transaction data, gas limit, and value, in the applicable networks, and as a result also in the post state. VM tests support only a single expectation with no specified network and no transaction (here, its role is played by `execution()`).

- `post_state` is a list of state definition in the same form as expected by `pre_state()`. State items that are not set explicitly default to their pre state.
- **networks** defines the forks under which the expectation is applicable. It should be a sublist of the following identifiers (also available in `ALL_FORKS`):
 - "Frontier"
 - "Homestead"
 - "EIP150"
 - "EIP158"
 - "Byzantium"
- `transaction` is a dictionary coming in two variants. For the main shard:

key	description
"data"	the transaction data,
"gasLimit"	the transaction gas limit,
"gasPrice"	the gas price,
"nonce"	the transaction nonce,
"value"	the transaction value

In addition, one should specify either the signature itself (via keys "v", "r", and "s") or a private key used for signing (via "secretKey").

3.6.7 Virtual Machine

Computation

BaseComputation

```
class eth.vm.computation.BaseComputation (state: eth.abc.StateAPI, message:
                                         eth.abc.MessageAPI, transaction_context:
                                         eth.abc.TransactionContextAPI)
```

The base class for all execution computations.

Note: Each *BaseComputation* class must be configured with:

`opcodes`: A mapping from the opcode integer value to the logic function for the opcode.

`_precompiles`: A mapping of contract address to the precompile function for execution of pre-compiled contracts.

add_child_computation (*child_computation*: eth.abc.ComputationAPI) → None
Add the given *child_computation*.

add_log_entry (*account*: NewType.<locals>.new_type, *topics*: Tuple[int, ...], *data*: bytes) → None
Add a log entry.

apply_child_computation (*child_msg*: eth.abc.MessageAPI) → eth.abc.ComputationAPI
Apply the vm message *child_msg* as a child computation.

classmethod apply_computation (*state*: eth.abc.StateAPI, *message*: eth.abc.MessageAPI,
 transaction_context: eth.abc.TransactionContextAPI) →
 eth.abc.ComputationAPI
Perform the computation that would be triggered by the VM message.

apply_create_message () → eth.abc.ComputationAPI
Execution of a VM message to create a new contract.

apply_message () → eth.abc.ComputationAPI
Execution of a VM message.

consume_gas (*amount*: int, *reason*: str) → None
Consume *amount* of gas from the remaining gas. Raise *eth.exceptions.OutOfGas* if there is not enough gas remaining.

extend_memory (*start_position*: int, *size*: int) → None
Extend the size of the memory to be at minimum *start_position* + *size* bytes in length. Raise *eth.exceptions.OutOfGas* if there is not enough gas to pay for extending the memory.

generate_child_computation (*child_msg*: eth.abc.MessageAPI) → eth.abc.ComputationAPI
Generate a child computation from the given *child_msg*.

get_accounts_for_deletion () → Tuple[Tuple[NewType.<locals>.new_type, New-
 Type.<locals>.new_type], ...]
Return a tuple of addresses that are registered for deletion.

get_gas_meter () → eth.abc.GasMeterAPI
Return the *GasMeterAPI* of the computation.

get_gas_refund () → int
Return the number of refunded gas.

get_gas_remaining () → int
Return the number of remaining gas.

get_gas_used () → int
Return the number of used gas.

get_log_entries () → Tuple[Tuple[bytes, Tuple[int, ...], bytes], ...]

Return the log entries for this computation and its children.

They are sorted in the same order they were emitted during the transaction processing, and include the sequential counter as the first element of the tuple representing every entry.

get_opcode_fn (*opcode: int*) → eth.abc.OpcodAPI

Return the function for the given opcode.

get_raw_log_entries () → Tuple[Tuple[int, bytes, Tuple[int, ...], bytes], ...]

Return a tuple of raw log entries.

memory_read (*start_position: int, size: int*) → memoryview

Read and return a view of *size* bytes from memory starting at *start_position*.

memory_read_bytes (*start_position: int, size: int*) → bytes

Read and return *size* bytes from memory starting at *start_position*.

memory_write (*start_position: int, size: int, value: bytes*) → None

Write *value* to memory at *start_position*. Require that `len(value) == size`.

prepare_child_message (*gas: int, to: NewType.<locals>.new_type, value: int, data: Union[bytes, memoryview], code: bytes, **kwargs*) → eth.abc.MessageAPI

Helper method for creating a child computation.

raise_if_error () → None

If there was an error during computation, raise it as an exception immediately.

Raises *VMEError* -

refund_gas (*amount: int*) → None

Add amount of gas to the pool of gas marked to be refunded.

register_account_for_deletion (*beneficiary: NewType.<locals>.new_type*) → None

Register the address of beneficiary for deletion.

return_gas (*amount: int*) → None

Return amount of gas to the available gas pool.

stack_dup (*position: int*) → None

Duplicate the stack item at *position* and pushes it onto the stack.

stack_swap (*position: int*) → None

Swap the item on the top of the stack with the item at *position*.

error

Return the *VMEError* of the computation. Raise `AttributeError` if no error exists.

is_error

Return `True` if the computation resulted in an error.

is_origin_computation

Return `True` if this computation is the outermost computation at `depth == 0`.

is_success

Return `True` if the computation did not result in an error.

output

Get the return value of the computation.

precompiles

Return a dictionary where the keys are the addresses of precompiles and the values are the precompile functions.

should_burn_gas

Return `True` if the remaining gas should be burned.

should_erase_return_data

Return `True` if the return data should be zeroed out due to an error.

should_return_gas

Return `True` if the remaining gas should be returned.

CodeStream

class `eth.vm.code_stream.CodeStream` (*code_bytes: bytes*)

is_valid_opcode (*position: int*) → `bool`

Return `True` if a valid opcode exists at `position`.

peek () → `int`

Return the ordinal value of the byte at the current program counter.

read (*size: int*) → `bytes`

Read and return the code from the current position of the cursor up to `size`.

seek (*program_counter: int*) → `Iterator[CodeStream]`

Return a `ContextManager` with the program counter set to `program_counter`.

ExecutionContext

class `eth.vm.execution_context.ExecutionContext` (*coinbase: [NewType](#).<locals>.new_type*, *timestamp: int*, *block_number: [NewType](#).<locals>.new_type*, *difficulty: int*, *gas_limit: int*, *prev_hashes: [Iterable](#)[[NewType](#).<locals>.new_type]*, *chain_id: int*)

block_number

Return the number of the block.

chain_id

Return the id of the chain.

coinbase

Return the coinbase address of the block.

difficulty

Return the difficulty of the block.

gas_limit

Return the gas limit of the block.

prev_hashes

Return an iterable of block hashes that precede the block.

timestamp

Return the timestamp of the block.

GasMeter

```
class eth.vm.gas_meter.GasMeter (start_gas: int, refund_strategy: Callable[[int, int], int] =  
    <function default_refund_strategy>)
```

```
    consume_gas (amount: int, reason: str) → None  
        Consume amount of gas for a defined reason.
```

```
    refund_gas (amount: int) → None  
        Refund amount of gas.
```

```
    return_gas (amount: int) → None  
        Return amount of gas.
```

Memory

```
class eth.vm.memory.Memory
```

```
    extend (start_position: int, size: int) → None  
        Extend the memory from the given start_position to the provided size.
```

```
    read (start_position: int, size: int) → memoryview  
        Return a view into the memory
```

```
    read_bytes (start_position: int, size: int) → bytes  
        Read a value from memory and return a fresh bytes instance
```

```
    write (start_position: int, size: int, value: bytes) → None  
        Write value into memory.
```

Message

```
class eth.vm.message.Message (gas: int, to: NewType.<locals>.new_type, sender: New-  
    Type.<locals>.new_type, value: int, data: Union[bytes,  
    memoryview], code: bytes, depth: int = 0, create_address:  
    NewType.<locals>.new_type = None, code_address: New-  
    Type.<locals>.new_type = None, should_transfer_value: bool =  
    True, is_static: bool = False)
```

Opcodes

```
class eth.vm.opcode.Opcodes
```

```
    classmethod as_opcode (logic_fn: Callable[..., Any], mnemonic: str, gas_cost: int) → T  
        Class factory method for turning vanilla functions into Opcode classes.
```

VM

VM

```
class eth.vm.base.VM(header: eth.abc.BlockHeaderAPI, chaindb: eth.abc.ChainDatabaseAPI,  
chain_context: eth.abc.ChainContextAPI, consensus_context:  
eth.abc.ConsensusContextAPI)
```

consensus_class

alias of `eth.consensus.pow.PowConsensus`

```
apply_all_transactions (transactions: Sequence[eth.abc.SignedTransactionAPI], base_header:  
eth.abc.BlockHeaderAPI) → Tuple[eth.abc.BlockHeaderAPI, Tu-  
ple[eth.abc.ReceiptAPI, ...], Tuple[eth.abc.ComputationAPI, ...]
```

Determine the results of applying all transactions to the base header. This does *not* update the current block or header of the VM.

Parameters

- **transactions** – an iterable of all transactions to apply
- **base_header** – the starting header to apply transactions to

Returns the final header, the receipts of each transaction, and the computations

```
apply_transaction (header: eth.abc.BlockHeaderAPI, transaction:  
eth.abc.SignedTransactionAPI) → Tuple[eth.abc.ReceiptAPI,  
eth.abc.ComputationAPI]
```

Apply the transaction to the current block. This is a wrapper around `apply_transaction()` with some extra orchestration logic.

Parameters

- **header** – header of the block before application
- **transaction** – to apply

```
classmethod build_state (db: eth.abc.AtomicDatabaseAPI, header: eth.abc.BlockHeaderAPI,  
chain_context: eth.abc.ChainContextAPI, previous_hashes: Iter-  
able[NewType.<locals>.new_type] = ()) → eth.abc.StateAPI
```

You probably want `VM().state` instead of this.

Occasionally, you want to build custom state against a particular header and DB, even if you don't have the VM initialized. This is a convenience method to do that.

```
classmethod create_execution_context (header: eth.abc.BlockHeaderAPI, prev_hashes:  
Iterable[NewType.<locals>.new_type],  
chain_context: eth.abc.ChainContextAPI)  
→ eth.abc.ExecutionContextAPI
```

Create and return the `ExecutionContextAPI` for the given header, iterable of block hashes that precede the block and the `chain_context`.

```
create_transaction (*args, **kwargs) → eth.abc.SignedTransactionAPI
```

Proxy for instantiating a signed transaction for this VM.

```
classmethod create_unsigned_transaction (*, nonce: int, gas_price: int, gas:  
int, to: NewType.<locals>.new_type,  
value: int, data: bytes) →  
eth.abc.UnsignedTransactionAPI
```

Proxy for instantiating an unsigned transaction for this VM.

```
execute_bytecode (origin: NewType.<locals>.new_type, gas_price: int, gas: int, to: New-  
Type.<locals>.new_type, sender: NewType.<locals>.new_type, value: int, data:  
bytes, code: bytes, code_address: NewType.<locals>.new_type = None) →  
eth.abc.ComputationAPI
```

Execute raw bytecode in the context of the current state of the virtual machine.

finalize_block (*block*: *eth.abc.BlockAPI*) → *eth.abc.BlockAPI*

Perform any finalization steps like awarding the block mining reward, and persisting the final state root.

classmethod generate_block_from_parent_header_and_coinbase (*parent_header*: *eth.abc.BlockHeaderAPI*, *coinbase*: *New-Type.<locals>.new_type*) → *eth.abc.BlockAPI*

Generate block from parent header and coinbase.

get_block () → *eth.abc.BlockAPI*

Return the current block.

classmethod get_block_class () → *Type[eth.abc.BlockAPI]*

Return the `Block` class that this VM uses for blocks.

get_header () → *eth.abc.BlockHeaderAPI*

Return the current header.

classmethod get_state_class () → *Type[eth.abc.StateAPI]*

Return the class that this VM uses for states.

classmethod get_transaction_class () → *Type[eth.abc.SignedTransactionAPI]*

Return the class that this VM uses for transactions.

import_block (*block*: *eth.abc.BlockAPI*) → *eth.abc.BlockAPI*

Import the given block to the chain.

mine_block (**args*, ***kwargs*) → *eth.abc.BlockAPI*

Mine the current block. Proxies to `self.pack_block` method.

pack_block (*block*: *eth.abc.BlockAPI*, **args*, ***kwargs*) → *eth.abc.BlockAPI*

Pack block for mining.

Parameters

- **coinbase** (*bytes*) – 20-byte public address to receive block reward
- **uncles_hash** (*bytes*) – 32 bytes
- **state_root** (*bytes*) – 32 bytes
- **transaction_root** (*bytes*) – 32 bytes
- **receipt_root** (*bytes*) – 32 bytes
- **bloom** (*int*) –
- **gas_used** (*int*) –
- **extra_data** (*bytes*) – 32 bytes
- **mix_hash** (*bytes*) – 32 bytes
- **nonce** (*bytes*) – 8 bytes

set_block_transactions (*base_block*: *eth.abc.BlockAPI*, *new_header*: *eth.abc.BlockHeaderAPI*, *transactions*: *Sequence[eth.abc.SignedTransactionAPI]*, *receipts*: *Sequence[eth.abc.ReceiptAPI]*) → *eth.abc.BlockAPI*

Create a new block with the given transactions.

state_in_temp_block () → *Iterator[eth.abc.StateAPI]*

Return a `ContextManager` with the current state wrapped in a temporary block.

validate_block (*block: eth.abc.BlockAPI*) → None

Validate the the given block.

classmethod validate_header (*header: eth.abc.BlockHeaderAPI, parent_header: eth.abc.BlockHeaderAPI*) → None

Raises **eth.exceptions.ValidationError** – if the header is not valid

classmethod validate_receipt (*receipt: eth.abc.ReceiptAPI*) → None

Validate the given receipt.

validate_seal (*header: eth.abc.BlockHeaderAPI*) → None

Validate the seal on the given header.

validate_seal_extension (*header: eth.abc.BlockHeaderAPI, parents: Iterable[eth.abc.BlockHeaderAPI]*) → None

Validate the seal on the given header when all parents must be present. Parent headers that are not yet in the database must be passed as *parents*.

classmethod validate_uncle (*block: eth.abc.BlockAPI, uncle: eth.abc.BlockAPI, uncle_parent: eth.abc.BlockAPI*) → None

Validate the given uncle in the context of the given block.

previous_hashes

Convenience API for accessing the previous 255 block hashes.

state

Return the current state.

Stack

class `eth.vm.stack.Stack`

VM Stack

dup (*position: int*) → None

Perform a DUP operation on the stack.

pop1_any () → Union[int, bytes]

Pop and return an element from the stack. The type of each element will be int or bytes, depending on whether it was pushed with `push_bytes` or `push_int`.

Raise `eth.exceptions.InsufficientStack` if the stack was empty.

pop1_bytes () → bytes

Pop and return a bytes element from the stack.

Raise `eth.exceptions.InsufficientStack` if the stack was empty.

pop1_int () → int

Pop and return an integer from the stack.

Raise `eth.exceptions.InsufficientStack` if the stack was empty.

pop_any (*num_items: int*) → Tuple[Union[int, bytes], ...]

Pop and return a tuple of items of length `num_items` from the stack. The type of each element will be int or bytes, depending on whether it was pushed with `stack_push_bytes` or `stack_push_int`.

Raise `eth.exceptions.InsufficientStack` if there are not enough items on the stack.

Items are ordered with the top of the stack as the first item in the tuple.

pop_bytes (*num_items: int*) → Tuple[bytes, ...]

Pop and return a tuple of bytes of length `num_items` from the stack.

Raise *eth.exceptions.InsufficientStack* if there are not enough items on the stack.

Items are ordered with the top of the stack as the first item in the tuple.

pop_ints (*num_items: int*) → Tuple[int, ...]

Pop and return a tuple of integers of length *num_items* from the stack.

Raise *eth.exceptions.InsufficientStack* if there are not enough items on the stack.

Items are ordered with the top of the stack as the first item in the tuple.

push_bytes (*value: bytes*) → None

Push a bytes item onto the stack.

push_int (*value: int*) → None

Push an integer item onto the stack.

swap (*position: int*) → None

Perform a SWAP operation on the stack.

State

BaseState

```
class eth.vm.state.BaseState (db: eth.abc.AtomicDatabaseAPI, execution_context: eth.abc.ExecutionContextAPI, state_root: bytes)
```

account_exists (*address: NewType.<locals>.new_type*) → bool

Return True if an account exists at address.

account_is_empty (*address: NewType.<locals>.new_type*) → bool

Return True if the account at address is empty, otherwise False.

commit (*snapshot: Tuple[NewType.<locals>.new_type, uuid.UUID]*) → None

Commit the journal to the point where the snapshot was taken. This merges in any changes that were recorded since the snapshot.

costless_execute_transaction (*transaction: eth.abc.SignedTransactionAPI*) → eth.abc.ComputationAPI

Execute the given *transaction* with a gas price of 0.

delete_account (*address: NewType.<locals>.new_type*) → None

Delete the account at the given address.

delete_code (*address: NewType.<locals>.new_type*) → None

Delete the code at address.

delete_storage (*address: NewType.<locals>.new_type*) → None

Delete the storage at address

delta_balance (*address: NewType.<locals>.new_type, delta: int*) → None

Apply *delta* to the balance at address.

classmethod get_account_db_class () → Type[eth.abc.AccountDatabaseAPI]

Return the *AccountDatabaseAPI* class that the state class uses.

get_ancestor_hash (*block_number: int*) → NewType.<locals>.new_type

Return the hash for the ancestor block with number *block_number*. Return the empty bytestring `b''` if the block number is outside of the range of available block numbers (typically the last 255 blocks).

get_balance (*address: NewType.<locals>.new_type*) → int

Return the balance for the account at address.

get_code (*address: NewType.<locals>.new_type*) → bytes
Return the code at address.

get_code_hash (*address: NewType.<locals>.new_type*) → NewType.<locals>.new_type
Return the hash of the code at address.

get_computation (*message: eth.abc.MessageAPI, transaction_context: eth.abc.TransactionContextAPI*) → eth.abc.ComputationAPI
Return a computation instance for the given *message* and *transaction_context*

get_nonce (*address: NewType.<locals>.new_type*) → int
Return the nonce at address.

get_storage (*address: NewType.<locals>.new_type, slot: int, from_journal: bool = True*) → int
Return the storage at *slot* for address.

classmethod get_transaction_context (*transaction: eth.abc.SignedTransactionAPI*) → eth.abc.TransactionContextAPI
Return the *TransactionContextAPI* for the given *transaction*

classmethod get_transaction_context_class () → Type[eth.abc.TransactionContextAPI]
Return the *BaseTransactionContext* class that the state class uses.

get_transaction_executor () → eth.abc.TransactionExecutorAPI
Return the transaction executor.

has_code_or_nonce (*address: NewType.<locals>.new_type*) → bool
Return True if either a nonce or code exists at the given address.

increment_nonce (*address: NewType.<locals>.new_type*) → None
Increment the nonce at address.

lock_changes () → None
Locks in all changes to state, typically just as a transaction starts.

This is used, for example, to look up the storage value from the start of the transaction, when calculating gas costs in EIP-2200: net gas metering.

make_state_root () → NewType.<locals>.new_type
Create and return the state root.

override_transaction_context (*gas_price: int*) → Iterator[None]
Return a *ContextManager* that overwrites the current transaction context, applying the given *gas_price*.

persist () → None
Persist the current state to the database.

revert (*snapshot: Tuple[NewType.<locals>.new_type, uuid.UUID]*) → None
Revert the VM to the state at the snapshot

set_balance (*address: NewType.<locals>.new_type, balance: int*) → None
Set *balance* to the balance at address.

set_code (*address: NewType.<locals>.new_type, code: bytes*) → None
Set code as the new code at address.

set_nonce (*address: NewType.<locals>.new_type, nonce: int*) → None
Set nonce as the new nonce at address.

set_storage (*address: NewType.<locals>.new_type, slot: int, value: int*) → None
Write *value* to the given *slot* at address.

snapshot () → Tuple[NewType.<locals>.new_type, uuid.UUID]
Perform a full snapshot of the current state.

Snapshots are a combination of the `state_root` at the time of the snapshot and the checkpoint from the journaled DB.

touch_account (*address: NewType.<locals>.new_type*) → None
Touch the account at the given address.

block_number
Return the current `block_number` from the current `execution_context`

coinbase
Return the current `coinbase` from the current `execution_context`

difficulty
Return the current `difficulty` from the current `execution_context`

gas_limit
Return the current `gas_limit` from the current `transaction_context`

logger
Return the logger.

state_root
Return the current `state_root` from the underlying database

timestamp
Return the current `timestamp` from the current `execution_context`

BaseTransactionExecutor

```
class eth.vm.state.BaseTransactionExecutor (vm_state: eth.abc.StateAPI)
```

BaseTransactionContext

```
class eth.vm.transaction_context.BaseTransactionContext (gas_price: int,
                                                         origin: NewType.<locals>.new_type)
```

get_next_log_counter () → int
Increment and return the log counter.

gas_price
Return the gas price of the transaction context.

origin
Return the origin of the transaction context.

Forks

Frontier

FrontierVM

```
class eth.vm.forks.frontier.FrontierVM(header: eth.abc.BlockHeaderAPI, chaindb:
    eth.abc.ChainDatabaseAPI, chain_context:
    eth.abc.ChainContextAPI, consensus_context:
    eth.abc.ConsensusContextAPI)
```

block_class

alias of `eth.vm.forks.frontier.blocks.FrontierBlock`

```
add_receipt_to_header(old_header: eth.abc.BlockHeaderAPI, receipt: eth.abc.ReceiptAPI) →
    eth.abc.BlockHeaderAPI
```

Apply the receipt to the old header, and return the resulting header. This may have storage-related side-effects. For example, pre-Byzantium, the state root hash is included in the receipt, and so must be stored into the database.

```
static compute_difficulty(parent_header: eth.rlp.headers.BlockHeader, timestamp: int) →
    int
```

Computes the difficulty for a frontier block based on the parent block.

```
static get_block_reward() → int
```

Return the amount in **wei** that should be given to a miner as a reward for this block.

Note: This is an abstract method that must be implemented in subclasses

```
classmethod get_nephew_reward() → int
```

Return the reward which should be given to the miner of the given *nephew*.

Note: This is an abstract method that must be implemented in subclasses

```
static get_uncle_reward(block_number: int, uncle: eth.abc.BlockAPI) → int
```

Return the reward which should be given to the miner of the given *uncle*.

Note: This is an abstract method that must be implemented in subclasses

```
static make_receipt(base_header: eth.abc.BlockHeaderAPI, transaction:
    eth.abc.SignedTransactionAPI, computation: eth.abc.ComputationAPI,
    state: eth.abc.StateAPI) → eth.abc.ReceiptAPI
```

Generate the receipt resulting from applying the transaction.

Parameters

- **base_header** – the header of the block before the transaction was applied.
- **transaction** – the transaction used to generate the receipt
- **computation** – the result of running the transaction computation
- **state** – the resulting state, after executing the computation

Returns receipt

FrontierState

```
class eth.vm.forks.frontier.state.FrontierState (db:      eth.abc.AtomicDatabaseAPI,
                                                execution_context:
                                                eth.abc.ExecutionContextAPI,
                                                state_root: bytes)
```

account_db_class
alias of `eth.db.account.AccountDB`

computation_class
alias of `eth.vm.forks.frontier.computation.FrontierComputation`

transaction_context_class
alias of `eth.vm.forks.frontier.transaction_context.FrontierTransactionContext`

transaction_executor_class
alias of `FrontierTransactionExecutor`

apply_transaction (*transaction*: `eth.abc.SignedTransactionAPI`) → `eth.abc.ComputationAPI`
Apply transaction to the vm state

Parameters **transaction** – the transaction to apply

Returns the computation

validate_transaction (*transaction*: `eth.abc.SignedTransactionAPI`) → None
Validate the given transaction.

FrontierComputation

```
class eth.vm.forks.frontier.computation.FrontierComputation (state:
                                                            eth.abc.StateAPI,
                                                            message:
                                                            eth.abc.MessageAPI,
                                                            transaction_context:
                                                            eth.abc.TransactionContextAPI)
```

A class for all execution computations in the Frontier fork. Inherits from `BaseComputation`

apply_create_message () → `eth.abc.ComputationAPI`
Execution of a VM message to create a new contract.

apply_message () → `eth.abc.ComputationAPI`
Execution of a VM message.

Homestead

HomesteadVM

```
class eth.vm.forks.homestead.HomesteadVM (header:  eth.abc.BlockHeaderAPI,  chaindb:
                                                eth.abc.ChainDatabaseAPI,  chain_context:
                                                eth.abc.ChainContextAPI,  consensus_context:
                                                eth.abc.ConsensusContextAPI)
```

block_class
alias of `eth.vm.forks.homestead.blocks.HomesteadBlock`

static compute_difficulty (*parent_header*: *eth.rlp.headers.BlockHeader*, *timestamp*: *int*) → *int*
 Computes the difficulty for a homestead block based on the parent block.

HomesteadState

class *eth.vm.forks.homestead.state.HomesteadState* (*db*: *eth.abc.AtomicDatabaseAPI*,
execution_context:
eth.abc.ExecutionContextAPI,
state_root: *bytes*)

computation_class

alias of *eth.vm.forks.homestead.computation.HomesteadComputation*

validate_transaction (*transaction*: *eth.abc.SignedTransactionAPI*) → *None*
 Validate the given *transaction*.

HomesteadComputation

class *eth.vm.forks.homestead.computation.HomesteadComputation* (*state*:
eth.abc.StateAPI,
message:
eth.abc.MessageAPI,
transaction_context:
eth.abc.TransactionContextAPI)

A class for all execution computations in the Frontier fork. Inherits from *FrontierComputation*

apply_create_message () → *eth.abc.ComputationAPI*
 Execution of a VM message to create a new contract.

TangerineWhistle

TangerineWhistleVM

class *eth.vm.forks.tangerine_whistle.TangerineWhistleVM* (*header*:
eth.abc.BlockHeaderAPI,
chaindb:
eth.abc.ChainDatabaseAPI,
chain_context:
eth.abc.ChainContextAPI,
consensus_context:
eth.abc.ConsensusContextAPI)

TangerineWhistleState

```

class eth.vm.forks.tangerine_whistle.state.TangerineWhistleState (db:
    eth.abc.AtomicDatabaseAPI,
    execu-
    tion_context:
    eth.abc.ExecutionContextAPI,
    state_root:
    bytes)

    computation_class
    alias of eth.vm.forks.tangerine_whistle.computation.
    TangerineWhistleComputation

```

TangerineWhistleComputation

```

class eth.vm.forks.tangerine_whistle.computation.TangerineWhistleComputation (state:
    eth.abc.StateAPI,
    mes-
    sage:
    eth.abc.MessageAPI,
    trans-
    ac-
    tion_context:
    eth.abc.TransactionAPI)

    A class for all execution computations in the TangerineWhistle fork. Inherits from
    HomesteadComputation

```

SpuriousDragon

SpuriousDragonVM

```

class eth.vm.forks.spurious_dragon.SpuriousDragonVM (header:
    eth.abc.BlockHeaderAPI,
    chaindb:
    eth.abc.ChainDatabaseAPI,
    chain_context:
    eth.abc.ChainContextAPI,
    consensus_context:
    eth.abc.ConsensusContextAPI)

    block_class
    alias of eth.vm.forks.spurious_dragon.blocks.SpuriousDragonBlock

```

SpuriousDragonState

```

class eth.vm.forks.spurious_dragon.state.SpuriousDragonState (db:
    eth.abc.AtomicDatabaseAPI,
    execution_context:
    eth.abc.ExecutionContextAPI,
    state_root: bytes)

```

computation_classalias of `eth.vm.forks.spurious_dragon.computation.SpuriousDragonComputation`**transaction_executor_class**alias of `SpuriousDragonTransactionExecutor`

SpuriousDragonComputation

class `eth.vm.forks.spurious_dragon.computation.SpuriousDragonComputation` (*state:* `eth.abc.StateAPI`, *message:* `eth.abc.MessageAPI`, *transaction_context:* `eth.abc.TransactionContextAPI`)

A class for all execution computations in the SpuriousDragon fork. Inherits from `HomesteadComputation`

apply_create_message () → `eth.abc.ComputationAPI`

Execution of a VM message to create a new contract.

Byzantium

ByzantiumVM

class `eth.vm.forks.byzantium.ByzantiumVM` (*header:* `eth.abc.BlockHeaderAPI`, *chaindb:* `eth.abc.ChainDatabaseAPI`, *chain_context:* `eth.abc.ChainContextAPI`, *consensus_context:* `eth.abc.ConsensusContextAPI`)

block_classalias of `eth.vm.forks.byzantium.blocks.ByzantiumBlock`**add_receipt_to_header** (*old_header:* `eth.abc.BlockHeaderAPI`, *receipt:* `eth.abc.ReceiptAPI`) → `eth.abc.BlockHeaderAPI`

Apply the receipt to the old header, and return the resulting header. This may have storage-related side-effects. For example, pre-Byzantium, the state root hash is included in the receipt, and so must be stored into the database.

static get_block_reward () → `int`Return the amount in **wei** that should be given to a miner as a reward for this block.

Note: This is an abstract method that must be implemented in subclasses

static make_receipt (*base_header:* `eth.abc.BlockHeaderAPI`, *transaction:* `eth.abc.SignedTransactionAPI`, *computation:* `eth.abc.ComputationAPI`, *state:* `eth.abc.StateAPI`) → `eth.abc.ReceiptAPI`

Generate the receipt resulting from applying the transaction.

Parameters

- **base_header** – the header of the block before the transaction was applied.

- **transaction** – the transaction used to generate the receipt
- **computation** – the result of running the transaction computation
- **state** – the resulting state, after executing the computation

Returns receipt

classmethod validate_receipt (*receipt: eth.abc.ReceiptAPI*) → None
 Validate the given receipt.

compute_difficulty
<https://github.com/ethereum/EIPs/issues/100>

ByzantiumState

class eth.vm.forks.byzantium.state.**ByzantiumState** (*db: eth.abc.AtomicDatabaseAPI,*
execution_context:
eth.abc.ExecutionContextAPI,
state_root: bytes)

computation_class
 alias of *eth.vm.forks.byzantium.computation.ByzantiumComputation*

ByzantiumComputation

class eth.vm.forks.byzantium.computation.**ByzantiumComputation** (*state:*
eth.abc.StateAPI,
message:
eth.abc.MessageAPI,
transaction_context:
eth.abc.TransactionContextAPI)

A class for all execution computations in the Byzantium fork. Inherits from *SpuriousDragonComputation*

Constantinople

ConstantinopleVM

class eth.vm.forks.constantinople.**ConstantinopleVM** (*header: eth.abc.BlockHeaderAPI,*
chaindb:
eth.abc.ChainDatabaseAPI,
chain_context:
eth.abc.ChainContextAPI,
consensus_context:
eth.abc.ConsensusContextAPI)

block_class
 alias of *eth.vm.forks.constantinople.blocks.ConstantinopleBlock*

static get_block_reward () → int
 Return the amount in **wei** that should be given to a miner as a reward for this block.

Note: This is an abstract method that must be implemented in subclasses

compute_difficulty

<https://github.com/ethereum/EIPs/issues/100>

ConstantinopleState

```
class eth.vm.forks.constantinople.state.ConstantinopleState (db:
    eth.abc.AtomicDatabaseAPI,
    execution_context:
    eth.abc.ExecutionContextAPI,
    state_root: bytes)
```

computation_class

alias of `eth.vm.forks.constantinople.computation.ConstantinopleComputation`

ConstantinopleComputation

```
class eth.vm.forks.constantinople.computation.ConstantinopleComputation (state:
    eth.abc.StateAPI,
    message:
    eth.abc.MessageAPI,
    transaction_context:
    eth.abc.TransactionContextAPI)
```

A class for all execution computations in the Constantinople fork. Inherits from `ByzantiumComputation`

```
get_gas_meter () → eth.vm.gas_meter.GasMeter
    Return the GasMeterAPI of the computation.
```

Petersburg**PetersburgVM**

```
class eth.vm.forks.petersburg.PetersburgVM (header: eth.abc.BlockHeaderAPI, chaindb:
    eth.abc.ChainDatabaseAPI, chain_context:
    eth.abc.ChainContextAPI, consensus_context:
    eth.abc.ConsensusContextAPI)
```

block_class

alias of `eth.vm.forks.petersburg.blocks.PetersburgBlock`

```
static get_block_reward () → int
    Return the amount in wei that should be given to a miner as a reward for this block.
```

Note: This is an abstract method that must be implemented in subclasses

compute_difficulty
<https://github.com/ethereum/EIPs/issues/100>

PetersburgState

class `eth.vm.forks.petersburg.state.PetersburgState` (*db: eth.abc.AtomicDatabaseAPI, execution_context: eth.abc.ExecutionContextAPI, state_root: bytes*)

computation_class
alias of `eth.vm.forks.petersburg.computation.PetersburgComputation`

PetersburgComputation

class `eth.vm.forks.petersburg.computation.PetersburgComputation` (*state: eth.abc.StateAPI, message: eth.abc.MessageAPI, transaction_context: eth.abc.TransactionContextAPI*)

A class for all execution computations in the Petersburg fork. Inherits from `ByzantiumComputation`

Istanbul

IstanbulVM

class `eth.vm.forks.istanbul.IstanbulVM` (*header: eth.abc.BlockHeaderAPI, chaindb: eth.abc.ChainDatabaseAPI, chain_context: eth.abc.ChainContextAPI, consensus_context: eth.abc.ConsensusContextAPI*)

block_class
alias of `eth.vm.forks.istanbul.blocks.IstanbulBlock`

compute_difficulty
<https://github.com/ethereum/EIPs/issues/100>

IstanbulState

class `eth.vm.forks.istanbul.state.IstanbulState` (*db: eth.abc.AtomicDatabaseAPI, execution_context: eth.abc.ExecutionContextAPI, state_root: bytes*)

computation_class
alias of `eth.vm.forks.istanbul.computation.IstanbulComputation`

IstanbulComputation

```
class eth.vm.forks.istanbul.computation.PetersburgComputation (state:
    eth.abc.StateAPI,
    message:
    eth.abc.MessageAPI,
    transac-
    tion_context:
    eth.abc.TransactionContextAPI)
```

A class for all execution computations in the Petersburg fork. Inherits from *ByzantiumComputation*

3.7 Contributing

Thank you for your interest in contributing! We welcome all contributions no matter their size. Please read along to learn how to get started. If you get stuck, feel free to reach for help in our [Gitter channel](#).

3.7.1 Setting the stage

First we need to clone the Py-EVM repository. Py-EVM depends on a submodule of the common tests across all clients, so we need to clone the repo with the `--recursive` flag. Example:

```
$ git clone --recursive https://github.com/ethereum/py-evm.git
```

Optional: Often, the best way to guarantee a clean Python 3 environment is with `virtualenv`. If we don't have `virtualenv` installed already, we first need to install it via `pip`.

```
pip install virtualenv
```

Then, we can initialize a new virtual environment `venv`, like:

```
virtualenv -p python3 venv
```

This creates a new directory `venv` where packages are installed isolated from any other global packages.

To activate the virtual directory we have to *source* it

```
. venv/bin/activate
```

After we have activated our virtual environment, installing all dependencies that are needed to run, develop and test all code in this repository is as easy as:

```
pip install -e .[dev]
```

3.7.2 Running the tests

A great way to explore the code base is to run the tests.

We can run all tests with:

```
pytest tests
```

However, running the entire test suite does take a very long time so often we just want to run a subset instead, like:

```
pytest tests/core/padding-utils/test_padding.py
```

We can also install `tox` to run the full test suite which also covers things like testing the code against different Python versions, linting etc.

It is important to understand that each Pull Request must pass the full test suite as part of the CI check, hence it is often convenient to have `tox` installed locally as well.

3.7.3 Code Style

When multiple people are working on the same body of code, it is important that they write code that conforms to a similar style. It often doesn't matter as much which style, but rather that they conform to one style.

To ensure your contribution conforms to the style being used in this project, we encourage you to read our [style guide](#).

3.7.4 Type Hints

The code bases is transitioning to use [type hints](#). Type hints make it easy to prevent certain types of bugs, enable richer tooling and enhance the documentation, making the code easier to follow.

All new code is required to land with type hints with the exception of test code that is not expected to use type hints.

All parameters as well as the return type of defs are expected to be typed with the exception of `self` and `cls` as seen in the following example.

```
def __init__(self, wrapped_db: DatabaseAPI) -> None:
    self.wrapped_db = wrapped_db
    self.reset()
```

3.7.5 Documentation

Good documentation will lead to quicker adoption and happier users. Please check out our guide on [how to create documentation for the Python Ethereum ecosystem](#).

3.7.6 Pull Requests

It's a good idea to make pull requests early on. A pull request represents the start of a discussion, and doesn't necessarily need to be the final, finished submission.

GitHub's documentation for working on pull requests is [available here](#).

Once you've made a pull request take a look at the Circle CI build status in the GitHub interface and make sure all tests are passing. In general pull requests that do not pass the CI build yet won't get reviewed unless explicitly requested.

If the pull request introduces changes that should be reflected in the release notes, please add a *newsfragment* file as explained [here](https://github.com/ethereum/py-evm/blob/master/newsfragments/README.md)<<https://github.com/ethereum/py-evm/blob/master/newsfragments/README.md>>_

If possible, the change to the release notes file should be included in the commit that introduces the feature or bugfix.

3.7.7 Releasing

Final test before each release

Before releasing a new version, build and test the package that will be released:

```
git checkout master && git pull

make package

# Preview the upcoming release notes
towncrier --draft
```

Build the release notes

Before bumping the version number, build the release notes. You must include the part of the version to bump (see below), which changes how the version number will show in the release notes.

```
make notes bump=$$VERSION_PART_TO_BUMP$$
```

If there are any errors, be sure to re-run make notes until it works.

Push the release to github & pypi

After confirming that the release package looks okay, release a new version:

```
make release bump=$$VERSION_PART_TO_BUMP$$
```

Which version part to bump

The version format for this repo is {major}.{minor}.{patch} for stable, and {major}.{minor}.{patch}-{stage} . {devnum} for unstable (stage can be alpha or beta).

During a release, specify which part to bump, like `make release bump=minor` or `make release bump=devnum`.

If you are in a beta version, `make release bump=stage` will switch to a stable.

To issue an unstable version when the current version is stable, specify the new version explicitly, like `make release bump="--new-version 4.0.0-alpha.1 devnum"`

3.8 Code of Conduct

3.8.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

3.8.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

3.8.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

3.8.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

3.8.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at piper@pipermerriam.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

3.8.6 Attribution

This Code of Conduct is adapted from the [Contributor Covenant](https://www.contributor-covenant.org/version/1/4/code-of-conduct.html), version 1.4, available at <https://www.contributor-covenant.org/version/1/4/code-of-conduct.html>

e

`eth.exceptions`, 64

A

- Account (class in eth.rlp.accounts), 66
- account_db_class (eth.vm.forks.frontier.state.FrontierState attribute), 84
- account_exists() (eth.abc.AccountDatabaseAPI method), 40
- account_exists() (eth.abc.StateAPI method), 42
- account_exists() (eth.db.account.AccountDB method), 56
- account_exists() (eth.vm.state.BaseState method), 80
- account_has_code_or_nonce() (eth.abc.AccountDatabaseAPI method), 40
- account_has_code_or_nonce() (eth.db.account.AccountDB method), 56
- account_is_empty() (eth.abc.AccountDatabaseAPI method), 40
- account_is_empty() (eth.abc.StateAPI method), 42
- account_is_empty() (eth.db.account.AccountDB method), 56
- account_is_empty() (eth.vm.state.BaseState method), 80
- AccountDatabaseAPI (class in eth.abc), 40
- AccountDB (class in eth.db.account), 56
- AccountStorageDatabaseAPI (class in eth.abc), 39
- AccountStorageDB (class in eth.db.storage), 63
- add_child_computation() (eth.abc.ComputationAPI method), 37
- add_child_computation() (eth.vm.computation.BaseComputation method), 73
- add_log_entry() (eth.abc.ComputationAPI method), 37
- add_log_entry() (eth.vm.computation.BaseComputation method), 73
- add_receipt() (eth.abc.ChainDatabaseAPI method), 33
- add_receipt() (eth.db.chain.ChainDB method), 59
- add_receipt_to_header() (eth.abc.VirtualMachineAPI method), 44
- add_receipt_to_header() (eth.vm.forks.byzantium.ByzantiumVM method), 87
- add_receipt_to_header() (eth.vm.forks.frontier.FrontierVM method), 83
- add_transaction() (eth.abc.ChainDatabaseAPI method), 33
- add_transaction() (eth.db.chain.ChainDB method), 59
- apply_all_transactions() (eth.abc.VirtualMachineAPI method), 45
- apply_all_transactions() (eth.vm.base.VM method), 77
- apply_child_computation() (eth.abc.ComputationAPI method), 37
- apply_child_computation() (eth.vm.computation.BaseComputation method), 73
- apply_computation() (eth.abc.ComputationAPI class method), 37
- apply_computation() (eth.vm.computation.BaseComputation class method), 73
- apply_create_message() (eth.abc.ComputationAPI method), 37
- apply_create_message() (eth.vm.computation.BaseComputation method), 73
- apply_create_message() (eth.vm.forks.frontier.computation.FrontierComputation method), 84
- apply_create_message() (eth.vm.forks.homestead.computation.HomesteadComputation method), 85
- apply_create_message() (eth.vm.forks.spurious_dragon.computation.SpuriousDragonComputation method), 87
- apply_message() (eth.abc.ComputationAPI method), 37
- apply_message() (eth.vm.computation.BaseComputation method), 73
- apply_message() (eth.vm.forks.frontier.computation.FrontierComputation method), 84
- apply_to() (eth.db.diff.DBDiff method), 60
- apply_transaction() (eth.abc.MiningChainAPI method), 52
- apply_transaction() (eth.abc.StateAPI method), 42
- apply_transaction() (eth.abc.VirtualMachineAPI method), 45
- apply_transaction() (eth.chains.base.MiningChain method), 55
- apply_transaction() (eth.vm.base.VM method), 77

- apply_transaction() (eth.vm.forks.frontier.state.FrontierState method), 84
 - as_opcode() (eth.abc.OpcodeAPI class method), 34
 - as_opcode() (eth.vm.opcode.Opcode class method), 76
 - as_signed_transaction() (eth.abc.UnsignedTransactionAPI method), 30
 - at_block_number() (in module eth.tools.builder.chain), 70
 - atomic_batch() (eth.abc.AtomicDatabaseAPI method), 32
 - atomic_batch() (eth.db.atomic.AtomicDB method), 57
 - atomic_batch() (eth.db.backends.level.LevelDB method), 58
 - AtomicDatabaseAPI (class in eth.abc), 32
 - AtomicDB (class in eth.db.atomic), 57
 - AtomicDBWriteBatch (class in eth.db.atomic), 57
- ## B
- BaseBlock (class in eth.rlp.blocks), 66
 - BaseChain (class in eth.chains.base), 52
 - BaseComputation (class in eth.vm.computation), 73
 - BaseDB (class in eth.db.backends.base), 58
 - BaseState (class in eth.vm.state), 80
 - BaseTransaction (class in eth.rlp.transactions), 67
 - BaseTransactionAPI (class in eth.abc), 30
 - BaseTransactionContext (class in eth.vm.transaction_context), 82
 - BaseTransactionExecutor (class in eth.vm.state), 82
 - BaseTransactionFields (class in eth.rlp.transactions), 67
 - BaseTransactionMethods (class in eth.rlp.transactions), 67
 - BaseUnsignedTransaction (class in eth.rlp.transactions), 67
 - BatchDB (class in eth.db.batch), 58
 - block_class (eth.vm.forks.byzantium.ByzantiumVM attribute), 87
 - block_class (eth.vm.forks.constantinople.ConstantinopleVM attribute), 88
 - block_class (eth.vm.forks.frontier.FrontierVM attribute), 83
 - block_class (eth.vm.forks.homestead.HomesteadVM attribute), 84
 - block_class (eth.vm.forks.istanbul.IstanbulVM attribute), 90
 - block_class (eth.vm.forks.petersburg.PetersburgVM attribute), 89
 - block_class (eth.vm.forks.spurious_dragon.SpuriousDragonVM attribute), 86
 - block_number (eth.abc.ExecutionContextAPI attribute), 37
 - block_number (eth.abc.StateAPI attribute), 44
 - block_number (eth.vm.execution_context.ExecutionContext attribute), 75
 - block_number (eth.vm.state.BaseState attribute), 82
 - BlockAPI (class in eth.abc), 31
 - BlockHeader (class in eth.rlp.headers), 66
 - BlockHeaderAPI (class in eth.abc), 30
 - BlockNotFound, 64
 - build_block_with_transactions() (eth.abc.ChainAPI method), 49
 - build_block_with_transactions() (eth.chains.base.Chain method), 53
 - build_computation() (eth.abc.TransactionExecutorAPI method), 41
 - build_evm_message() (eth.abc.TransactionExecutorAPI method), 41
 - build_state() (eth.abc.VirtualMachineAPI class method), 45
 - build_state() (eth.vm.base.VM class method), 77
 - ByzantiumComputation (class in eth.vm.forks.byzantium.computation), 88
 - ByzantiumState (class in eth.vm.forks.byzantium.state), 88
 - ByzantiumVM (class in eth.vm.forks.byzantium), 87
- ## C
- CacheDB (class in eth.db.cache), 59
 - CanonicalHeadNotFound, 64
 - Chain (class in eth.chains.base), 53
 - chain_id (eth.abc.ExecutionContextAPI attribute), 37
 - chain_id (eth.vm.execution_context.ExecutionContext attribute), 75
 - chain_id() (in module eth.tools.builder.chain), 69
 - chain_split() (in module eth.tools.builder.chain), 69
 - ChainAPI (class in eth.abc), 49
 - ChainDatabaseAPI (class in eth.abc), 33
 - ChainDB (class in eth.db.chain), 59
 - chaindb_class (eth.chains.base.Chain attribute), 53
 - check_signature_validity() (eth.abc.SignedTransactionAPI method), 31
 - clear() (eth.db.batch.BatchDB method), 58
 - clear() (eth.db.journal.JournalDB method), 62
 - CodeStream (class in eth.vm.code_stream), 75
 - CodeStreamAPI (class in eth.abc), 36
 - coinbase (eth.abc.ExecutionContextAPI attribute), 37
 - coinbase (eth.abc.StateAPI attribute), 44
 - coinbase (eth.vm.execution_context.ExecutionContext attribute), 75
 - coinbase (eth.vm.state.BaseState attribute), 82
 - commit() (eth.abc.AccountDatabaseAPI method), 40
 - commit() (eth.abc.AccountStorageDatabaseAPI method), 39
 - commit() (eth.abc.StateAPI method), 42
 - commit() (eth.db.account.AccountDB method), 56
 - commit() (eth.db.storage.AccountStorageDB method), 63
 - commit() (eth.vm.state.BaseState method), 80
 - commit_to() (eth.db.storage.StorageLookup method), 64
 - computation_class (eth.vm.forks.byzantium.state.ByzantiumState attribute), 88

[computation_class](#) (eth.vm.forks.constantinople.state.ConstantinopleState attribute), 89
[computation_class](#) (eth.vm.forks.constantinople.state.ConstantinopleState static method), 45
[computation_class](#) (eth.vm.forks.frontier.state.FrontierState create_execution_context() (eth.vm.base.VM class attribute), 84 method), 77
[computation_class](#) (eth.vm.forks.homestead.state.HomesteadState create_header_from_parent() (eth.abc.ChainAPI attribute), 85 method), 49
[computation_class](#) (eth.vm.forks.istanbul.state.IstanbulState create_header_from_parent() (eth.abc.VirtualMachineAPI class attribute), 90 method), 45
[computation_class](#) (eth.vm.forks.petersburg.state.PetersburgState create_header_from_parent() (eth.chains.base.Chain attribute), 90 method), 53
[computation_class](#) (eth.vm.forks.spurious_dragon.state.SpuriousDragonState create_transaction() (eth.abc.ChainAPI method), 49 attribute), 86 create_transaction() (eth.abc.VirtualMachineAPI method), 45
[computation_class](#) (eth.vm.forks.tangerine_whistle.state.TangerineWhistleState create_transaction() (eth.chains.base.Chain attribute), 86 method), 53
[ComputationAPI](#) (class in eth.abc), 37 create_transaction() (eth.vm.base.VM method), 77
[compute_difficulty](#) (eth.vm.forks.byzantium.ByzantiumVM create_unsigned_transaction() (eth.abc.ChainAPI attribute), 88 method), 49
[compute_difficulty](#) (eth.vm.forks.constantinople.ConstantinopleVM create_unsigned_transaction() (eth.abc.SignedTransactionAPI class attribute), 89 method), 31
[compute_difficulty](#) (eth.vm.forks.istanbul.IstanbulVM attribute), 90 create_unsigned_transaction() (eth.abc.VirtualMachineAPI class attribute), 90 method), 45
[compute_difficulty](#) (eth.vm.forks.petersburg.PetersburgVM create_unsigned_transaction() (eth.chains.base.Chain attribute), 89 method), 53
[compute_difficulty\(\)](#) (eth.abc.VirtualMachineAPI class create_unsigned_transaction() (eth.vm.base.VM method), 45 method), 45
[compute_difficulty\(\)](#) (eth.vm.forks.frontier.FrontierVM create_unsigned_transaction() (eth.abc.VirtualMachineAPI class static method), 83 method), 77
[compute_difficulty\(\)](#) (eth.vm.forks.homestead.HomesteadVM create_unsigned_transaction() (eth.abc.VirtualMachineAPI class static method), 84 method), 45
[ConfigurableAPI](#) (class in eth.abc), 42
[configure_header\(\)](#) (eth.abc.VirtualMachineAPI method), 45
[consensus_class](#) (eth.vm.base.VM attribute), 77
[consensus_context_class](#) (eth.chains.base.Chain attribute), 53
[ConstantinopleComputation](#) (class in eth.vm.forks.constantinople.computation), 89
[ConstantinopleState](#) (class in eth.vm.forks.constantinople.state), 89
[ConstantinopleVM](#) (class in eth.vm.forks.constantinople), 88
[consume_gas\(\)](#) (eth.abc.ComputationAPI method), 37
[consume_gas\(\)](#) (eth.abc.GasMeterAPI method), 34
[consume_gas\(\)](#) (eth.vm.computation.BaseComputation method), 73
[consume_gas\(\)](#) (eth.vm.gas_meter.GasMeter method), 76
[ContractCreationCollision](#), 64
[copy\(\)](#) (eth.abc.BaseTransactionAPI method), 30
[copy\(\)](#) (in module eth.tools.builder.chain), 69
[costless_execute_transaction\(\)](#) (eth.abc.StateAPI method), 42
[costless_execute_transaction\(\)](#) (eth.vm.state.BaseState method), 80
[dao_fork_at\(\)](#) (in module eth.tools.builder.chain), 68
[DatabaseAPI](#) (class in eth.abc), 31
[DBDiff](#) (class in eth.db.diff), 60
[DBDiffTracker](#) (class in eth.db.diff), 61
[delete\(\)](#) (eth.abc.AccountStorageDatabaseAPI method), 39
[delete\(\)](#) (eth.abc.DatabaseAPI method), 31
[delete\(\)](#) (eth.db.backends.base.BaseDB method), 58
[delete\(\)](#) (eth.db.storage.AccountStorageDB method), 63
[delete_account\(\)](#) (eth.abc.AccountDatabaseAPI method), 40
[delete_account\(\)](#) (eth.abc.StateAPI method), 42
[delete_account\(\)](#) (eth.db.account.AccountDB method), 56
[delete_account\(\)](#) (eth.vm.state.BaseState method), 80
[delete_code\(\)](#) (eth.abc.AccountDatabaseAPI method), 40
[delete_code\(\)](#) (eth.abc.StateAPI method), 42
[delete_code\(\)](#) (eth.db.account.AccountDB method), 56
[delete_code\(\)](#) (eth.vm.state.BaseState method), 80
[delete_storage\(\)](#) (eth.abc.AccountDatabaseAPI method), 40
[delete_storage\(\)](#) (eth.abc.StateAPI method), 42
[delete_storage\(\)](#) (eth.db.account.AccountDB method), 56
[delete_storage\(\)](#) (eth.vm.state.BaseState method), 80
[deleted_keys\(\)](#) (eth.db.diff.DBDiff method), 60
[delta_balance\(\)](#) (eth.abc.StateAPI method), 42
[delta_balance\(\)](#) (eth.vm.state.BaseState method), 80

diff() (eth.db.journal.JournalDB method), 62
 difficulty (eth.abc.ExecutionContextAPI attribute), 37
 difficulty (eth.abc.StateAPI attribute), 44
 difficulty (eth.vm.execution_context.ExecutionContext attribute), 75
 difficulty (eth.vm.state.BaseState attribute), 82
 DiffMissingError (class in eth.db.diff), 61
 disable_dao_fork() (in module eth.tools.builder.chain), 68
 disable_pow_check() (in module eth.tools.builder.chain), 69
 discard() (eth.abc.AccountDatabaseAPI method), 40
 discard() (eth.abc.AccountStorageDatabaseAPI method), 39
 discard() (eth.db.account.AccountDB method), 56
 discard() (eth.db.journal.JournalDB method), 62
 discard() (eth.db.storage.AccountStorageDB method), 63
 dup() (eth.abc.StackAPI method), 35
 dup() (eth.vm.stack.Stack method), 79

E

enable_pow_mining() (in module eth.tools.builder.chain), 69
 ensure_header() (eth.chains.base.Chain method), 53
 error (eth.abc.ComputationAPI attribute), 39
 error (eth.vm.computation.BaseComputation attribute), 74
 estimate_gas() (eth.abc.ChainAPI method), 49
 estimate_gas() (eth.chains.base.Chain method), 53
 eth.exceptions (module), 64
 eth.vm.opcode.as_opcode() (built-in function), 28
 execute_bytecode() (eth.abc.VirtualMachineAPI method), 46
 execute_bytecode() (eth.vm.base.VM method), 77
 execution() (in module eth.tools.fixtures.fillers), 71
 ExecutionContext (class in eth.vm.execution_context), 75
 ExecutionContextAPI (class in eth.abc), 37
 exists() (eth.abc.ChainDatabaseAPI method), 33
 exists() (eth.abc.DatabaseAPI method), 32
 exists() (eth.db.backends.base.BaseDB method), 58
 exists() (eth.db.chain.ChainDB method), 59
 expect() (in module eth.tools.fixtures.fillers), 72
 extend() (eth.abc.MemoryAPI method), 35
 extend() (eth.vm.memory.Memory method), 76
 extend_memory() (eth.abc.ComputationAPI method), 38
 extend_memory() (eth.vm.computation.BaseComputation method), 73

F

finalize_block() (eth.abc.VirtualMachineAPI method), 46
 finalize_block() (eth.vm.base.VM method), 77
 finalize_computation() (eth.abc.TransactionExecutorAPI method), 41
 flatten() (eth.db.journal.JournalDB method), 62
 fork_at() (in module eth.tools.builder.chain), 68

from_base_transaction() (eth.abc.SignedTransactionAPI class method), 31
 from_base_transaction() (eth.rlp.transactions.BaseTransaction class method), 67
 from_genesis() (eth.abc.ChainAPI class method), 49
 from_genesis() (eth.chains.base.Chain class method), 53
 from_genesis_header() (eth.abc.ChainAPI class method), 49
 from_genesis_header() (eth.abc.HeaderChainAPI class method), 48
 from_genesis_header() (eth.chains.base.Chain class method), 53
 from_header() (eth.abc.BlockAPI class method), 31
 from_parent() (eth.rlp.headers.BlockHeader class method), 66
 FrontierComputation (class in eth.vm.forks.frontier.computation), 84
 FrontierState (class in eth.vm.forks.frontier.state), 84
 FrontierVM (class in eth.vm.forks.frontier), 83
 FullStack, 64

G

gas_limit (eth.abc.ExecutionContextAPI attribute), 37
 gas_limit (eth.abc.StateAPI attribute), 44
 gas_limit (eth.vm.execution_context.ExecutionContext attribute), 75
 gas_limit (eth.vm.state.BaseState attribute), 82
 gas_price (eth.abc.TransactionContextAPI attribute), 35
 gas_price (eth.vm.transaction_context.BaseTransactionContext attribute), 82
 gas_used_by() (eth.abc.BaseTransactionAPI method), 30
 gas_used_by() (eth.rlp.transactions.BaseTransactionMethods method), 67
 GasMeter (class in eth.vm.gas_meter), 76
 GasMeterAPI (class in eth.abc), 34
 generate_block_from_parent_header_and_coinbase() (eth.abc.VirtualMachineAPI class method), 46
 generate_block_from_parent_header_and_coinbase() (eth.vm.base.VM class method), 78
 generate_child_computation() (eth.abc.ComputationAPI method), 38
 generate_child_computation() (eth.vm.computation.BaseComputation method), 73
 genesis() (in module eth.tools.builder.chain), 69
 get() (eth.abc.AccountStorageDatabaseAPI method), 39
 get() (eth.abc.ChainDatabaseAPI method), 33
 get() (eth.db.chain.ChainDB method), 59
 get() (eth.db.storage.AccountStorageDB method), 63
 get_account_db_class() (eth.abc.StateAPI class method), 42
 get_account_db_class() (eth.vm.state.BaseState class method), 80

- [get_accounts_for_deletion\(\)](#) (eth.abc.ComputationAPI method), 38
[get_accounts_for_deletion\(\)](#) (eth.vm.computation.BaseComputation method), 73
[get_ancestor_hash\(\)](#) (eth.abc.StateAPI method), 43
[get_ancestor_hash\(\)](#) (eth.vm.state.BaseState method), 80
[get_ancestors\(\)](#) (eth.abc.ChainAPI method), 50
[get_ancestors\(\)](#) (eth.chains.base.Chain method), 53
[get_balance\(\)](#) (eth.abc.AccountDatabaseAPI method), 40
[get_balance\(\)](#) (eth.abc.StateAPI method), 43
[get_balance\(\)](#) (eth.db.account.AccountDB method), 56
[get_balance\(\)](#) (eth.vm.state.BaseState method), 80
[get_block\(\)](#) (eth.abc.ChainAPI method), 50
[get_block\(\)](#) (eth.abc.VirtualMachineAPI method), 46
[get_block\(\)](#) (eth.chains.base.Chain method), 53
[get_block\(\)](#) (eth.vm.base.VM method), 78
[get_block_by_hash\(\)](#) (eth.abc.ChainAPI method), 50
[get_block_by_hash\(\)](#) (eth.chains.base.Chain method), 54
[get_block_by_header\(\)](#) (eth.abc.ChainAPI method), 50
[get_block_by_header\(\)](#) (eth.chains.base.Chain method), 54
[get_block_class\(\)](#) (eth.abc.VirtualMachineAPI class method), 46
[get_block_class\(\)](#) (eth.vm.base.VM class method), 78
[get_block_header_by_hash\(\)](#) (eth.abc.ChainAPI method), 50
[get_block_header_by_hash\(\)](#) (eth.abc.HeaderChainAPI method), 48
[get_block_header_by_hash\(\)](#) (eth.abc.HeaderDatabaseAPI method), 32
[get_block_header_by_hash\(\)](#) (eth.chains.base.Chain method), 54
[get_block_header_by_hash\(\)](#) (eth.db.header.HeaderDB method), 61
[get_block_reward\(\)](#) (eth.abc.VirtualMachineAPI static method), 46
[get_block_reward\(\)](#) (eth.vm.forks.byzantium.ByzantiumVM static method), 87
[get_block_reward\(\)](#) (eth.vm.forks.constantinople.ConstantinopleVM static method), 88
[get_block_reward\(\)](#) (eth.vm.forks.frontier.FrontierVM static method), 83
[get_block_reward\(\)](#) (eth.vm.forks.petersburg.PetersburgVM static method), 89
[get_block_transaction_hashes\(\)](#) (eth.abc.ChainDatabaseAPI method), 33
[get_block_transaction_hashes\(\)](#) (eth.db.chain.ChainDB method), 59
[get_block_transactions\(\)](#) (eth.abc.ChainDatabaseAPI method), 33
[get_block_transactions\(\)](#) (eth.db.chain.ChainDB method), 59
[get_block_uncles\(\)](#) (eth.abc.ChainDatabaseAPI method), 33
[get_block_uncles\(\)](#) (eth.db.chain.ChainDB method), 59
[get_canonical_block_by_number\(\)](#) (eth.abc.ChainAPI method), 50
[get_canonical_block_by_number\(\)](#) (eth.chains.base.Chain method), 54
[get_canonical_block_hash\(\)](#) (eth.abc.ChainAPI method), 50
[get_canonical_block_hash\(\)](#) (eth.abc.HeaderChainAPI method), 48
[get_canonical_block_hash\(\)](#) (eth.abc.HeaderDatabaseAPI method), 32
[get_canonical_block_hash\(\)](#) (eth.chains.base.Chain method), 54
[get_canonical_block_hash\(\)](#) (eth.db.header.HeaderDB method), 61
[get_canonical_block_header_by_number\(\)](#) (eth.abc.ChainAPI method), 50
[get_canonical_block_header_by_number\(\)](#) (eth.abc.HeaderChainAPI method), 48
[get_canonical_block_header_by_number\(\)](#) (eth.abc.HeaderDatabaseAPI method), 32
[get_canonical_block_header_by_number\(\)](#) (eth.chains.base.Chain method), 54
[get_canonical_block_header_by_number\(\)](#) (eth.db.header.HeaderDB method), 61
[get_canonical_head\(\)](#) (eth.abc.ChainAPI method), 50
[get_canonical_head\(\)](#) (eth.abc.HeaderChainAPI method), 48
[get_canonical_head\(\)](#) (eth.abc.HeaderDatabaseAPI method), 32
[get_canonical_head\(\)](#) (eth.chains.base.Chain method), 54
[get_canonical_head\(\)](#) (eth.db.header.HeaderDB method), 61
[get_canonical_transaction\(\)](#) (eth.abc.ChainAPI method), 50
[get_canonical_transaction\(\)](#) (eth.chains.base.Chain method), 54
[get_canonical_transaction_by_index\(\)](#) (eth.abc.ChainAPI method), 50
[get_canonical_transaction_by_index\(\)](#) (eth.chains.base.Chain method), 54
[get_canonical_transaction_index\(\)](#) (eth.abc.ChainAPI method), 50
[get_canonical_transaction_index\(\)](#) (eth.chains.base.Chain method), 54
[get_chaindb_class\(\)](#) (eth.abc.ChainAPI class method), 50
[get_chaindb_class\(\)](#) (eth.chains.base.Chain class method), 54
[get_changed_root\(\)](#) (eth.abc.AccountStorageDatabaseAPI method), 39
[get_changed_root\(\)](#) (eth.db.storage.AccountStorageDB method), 63
[get_code\(\)](#) (eth.abc.AccountDatabaseAPI method), 40

- get_code() (eth.abc.StateAPI method), 43
- get_code() (eth.db.account.AccountDB method), 56
- get_code() (eth.vm.state.BaseState method), 80
- get_code_hash() (eth.abc.AccountDatabaseAPI method), 40
- get_code_hash() (eth.abc.StateAPI method), 43
- get_code_hash() (eth.db.account.AccountDB method), 56
- get_code_hash() (eth.vm.state.BaseState method), 81
- get_computation() (eth.abc.StateAPI method), 43
- get_computation() (eth.vm.state.BaseState method), 81
- get_gas_meter() (eth.abc.ComputationAPI method), 38
- get_gas_meter() (eth.vm.computation.BaseComputation method), 73
- get_gas_meter() (eth.vm.forks.constantinople.computation.ComputationAPI method), 89
- get_gas_refund() (eth.abc.ComputationAPI method), 38
- get_gas_refund() (eth.vm.computation.BaseComputation method), 73
- get_gas_remaining() (eth.abc.ComputationAPI method), 38
- get_gas_remaining() (eth.vm.computation.BaseComputation method), 73
- get_gas_used() (eth.abc.ComputationAPI method), 38
- get_gas_used() (eth.vm.computation.BaseComputation method), 73
- get_header() (eth.abc.VirtualMachineAPI method), 46
- get_header() (eth.vm.base.VM method), 78
- get_headerdb_class() (eth.abc.HeaderChainAPI class method), 48
- get_intrinsic_gas() (eth.abc.BaseTransactionAPI method), 30
- get_log_entries() (eth.abc.ComputationAPI method), 38
- get_log_entries() (eth.vm.computation.BaseComputation method), 73
- get_message_for_signing() (eth.abc.SignedTransactionAPI method), 31
- get_nephew_reward() (eth.abc.VirtualMachineAPI class method), 46
- get_nephew_reward() (eth.vm.forks.frontier.FrontierVM class method), 83
- get_next_log_counter() (eth.abc.TransactionContextAPI method), 35
- get_next_log_counter() (eth.vm.transaction_context.BaseTransactionContextAPI method), 82
- get_nonce() (eth.abc.AccountDatabaseAPI method), 40
- get_nonce() (eth.abc.StateAPI method), 43
- get_nonce() (eth.db.account.AccountDB method), 56
- get_nonce() (eth.vm.state.BaseState method), 81
- get_opcode_fn() (eth.abc.ComputationAPI method), 38
- get_opcode_fn() (eth.vm.computation.BaseComputation method), 74
- get_prev_hashes() (eth.abc.VirtualMachineAPI class method), 46
- get_raw_log_entries() (eth.abc.ComputationAPI method), 38
- get_raw_log_entries() (eth.vm.computation.BaseComputation method), 74
- get_receipt_by_index() (eth.abc.ChainDatabaseAPI method), 33
- get_receipt_by_index() (eth.db.chain.ChainDB method), 59
- get_receipts() (eth.abc.ChainDatabaseAPI method), 33
- get_receipts() (eth.db.chain.ChainDB method), 59
- get_score() (eth.abc.ChainAPI method), 51
- get_score() (eth.abc.HeaderDatabaseAPI method), 32
- get_score() (eth.chains.base.Chain method), 54
- get_score() (eth.chains.base.ChainAPI method), 61
- get_sender() (eth.abc.SignedTransactionAPI method), 31
- get_state_class() (eth.abc.VirtualMachineAPI class method), 46
- get_state_class() (eth.vm.base.VM class method), 78
- get_storage() (eth.abc.AccountDatabaseAPI method), 40
- get_storage() (eth.abc.StateAPI method), 43
- get_storage() (eth.db.account.AccountDB method), 56
- get_storage() (eth.vm.state.BaseState method), 81
- get_transaction_by_index() (eth.abc.ChainDatabaseAPI method), 33
- get_transaction_by_index() (eth.db.chain.ChainDB method), 59
- get_transaction_class() (eth.abc.BlockAPI class method), 31
- get_transaction_class() (eth.abc.VirtualMachineAPI class method), 46
- get_transaction_class() (eth.rlp.blocks.BaseBlock class method), 66
- get_transaction_class() (eth.vm.base.VM class method), 78
- get_transaction_context() (eth.abc.StateAPI class method), 43
- get_transaction_context() (eth.vm.state.BaseState class method), 81
- get_transaction_context_class() (eth.abc.StateAPI class method), 43
- get_transaction_context_class() (eth.vm.state.BaseState class method), 81
- get_transaction_executor() (eth.abc.StateAPI method), 43
- get_transaction_executor() (eth.vm.state.BaseState method), 81
- get_transaction_index() (eth.abc.ChainDatabaseAPI method), 34
- get_transaction_index() (eth.db.chain.ChainDB method), 60
- get_transaction_receipt() (eth.abc.ChainAPI method), 51
- get_transaction_receipt() (eth.chains.base.Chain method), 54
- get_transaction_receipt_by_index() (eth.abc.ChainAPI method), 51

- [get_transaction_receipt_by_index\(\)](#)
 (eth.chains.base.Chain method), 55
[get_transaction_result\(\)](#) (eth.abc.ChainAPI method), 51
[get_transaction_result\(\)](#) (eth.chains.base.Chain method), 55
[get_uncle_reward\(\)](#) (eth.abc.VirtualMachineAPI static method), 46
[get_uncle_reward\(\)](#) (eth.vm.forks.frontier.FrontierVM static method), 83
[get_vm\(\)](#) (eth.abc.ChainAPI method), 51
[get_vm\(\)](#) (eth.chains.base.Chain method), 55
[get_vm\(\)](#) (eth.chains.base.MiningChain method), 55
[get_vm_class\(\)](#) (eth.abc.ChainAPI class method), 51
[get_vm_class\(\)](#) (eth.chains.base.BaseChain class method), 52
[get_vm_class_for_block_number\(\)](#) (eth.abc.ChainAPI class method), 51
[get_vm_class_for_block_number\(\)](#) (eth.chains.base.BaseChain class method), 52
- ## H
- [Halt](#), 64
[has_changed_root](#) (eth.abc.AccountStorageDatabaseAPI attribute), 40
[has_changed_root](#) (eth.db.storage.AccountStorageDB attribute), 64
[has_code_or_nonce\(\)](#) (eth.abc.StateAPI method), 43
[has_code_or_nonce\(\)](#) (eth.vm.state.BaseState method), 81
[has_root\(\)](#) (eth.abc.AccountDatabaseAPI method), 41
[has_root\(\)](#) (eth.db.account.AccountDB method), 57
[hash](#) (eth.abc.BlockAPI attribute), 31
[header_exists\(\)](#) (eth.abc.HeaderChainAPI method), 48
[header_exists\(\)](#) (eth.abc.HeaderDatabaseAPI method), 32
[header_exists\(\)](#) (eth.db.header.HeaderDB method), 61
[HeaderChainAPI](#) (class in eth.abc), 48
[HeaderDatabaseAPI](#) (class in eth.abc), 32
[HeaderDB](#) (class in eth.db.header), 61
[HeaderNotFound](#), 64
[HomesteadComputation](#) (class in eth.vm.forks.homestead.computation), 85
[HomesteadState](#) (class in eth.vm.forks.homestead.state), 85
[HomesteadVM](#) (class in eth.vm.forks.homestead), 84
- ## I
- [import_block\(\)](#) (eth.abc.ChainAPI method), 51
[import_block\(\)](#) (eth.abc.VirtualMachineAPI method), 47
[import_block\(\)](#) (eth.chains.base.Chain method), 55
[import_block\(\)](#) (eth.chains.base.MiningChain method), 56
[import_block\(\)](#) (eth.vm.base.VM method), 78
[import_block\(\)](#) (in module eth.tools.builder.chain), 69
[import_blocks\(\)](#) (in module eth.tools.builder.chain), 69
[import_header\(\)](#) (eth.abc.HeaderChainAPI method), 48
[IncorrectContractCreationAddress](#), 65
[increment_nonce\(\)](#) (eth.abc.AccountDatabaseAPI method), 41
[increment_nonce\(\)](#) (eth.abc.StateAPI method), 43
[increment_nonce\(\)](#) (eth.db.account.AccountDB method), 57
[increment_nonce\(\)](#) (eth.vm.state.BaseState method), 81
[InsufficientFunds](#), 65
[InsufficientStack](#), 65
[intrinsic_gas](#) (eth.abc.BaseTransactionAPI attribute), 30
[intrinsic_gas](#) (eth.rlp.transactions.BaseTransactionMethods attribute), 67
[InvalidInstruction](#), 65
[InvalidJumpDestination](#), 65
[is_error](#) (eth.abc.ComputationAPI attribute), 39
[is_error](#) (eth.vm.computation.BaseComputation attribute), 74
[is_genesis](#) (eth.abc.BlockAPI attribute), 31
[is_genesis](#) (eth.rlp.blocks.BaseBlock attribute), 66
[is_origin_computation](#) (eth.abc.ComputationAPI attribute), 39
[is_origin_computation](#) (eth.vm.computation.BaseComputation attribute), 74
[is_signature_valid](#) (eth.abc.SignedTransactionAPI attribute), 31
[is_signature_valid](#) (eth.rlp.transactions.BaseTransaction attribute), 67
[is_success](#) (eth.abc.ComputationAPI attribute), 39
[is_success](#) (eth.vm.computation.BaseComputation attribute), 74
[is_valid_opcode\(\)](#) (eth.abc.CodeStreamAPI method), 36
[is_valid_opcode\(\)](#) (eth.vm.code_stream.CodeStream method), 75
[IstanbulState](#) (class in eth.vm.forks.istanbul.state), 90
[IstanbulVM](#) (class in eth.vm.forks.istanbul), 90
- ## J
- [join\(\)](#) (eth.db.diff.DBDiff class method), 60
[JournalDB](#) (class in eth.db.journal), 62
- ## L
- [LevelDB](#) (class in eth.db.backends.level), 58
[lock_changes\(\)](#) (eth.abc.AccountStorageDatabaseAPI method), 39
[lock_changes\(\)](#) (eth.abc.StateAPI method), 43
[lock_changes\(\)](#) (eth.db.storage.AccountStorageDB method), 63
[lock_changes\(\)](#) (eth.vm.state.BaseState method), 81
[Log](#) (class in eth.rlp.logs), 67
[LogAPI](#) (class in eth.abc), 30
[logger](#) (eth.abc.StateAPI attribute), 44
[logger](#) (eth.vm.state.BaseState attribute), 82

M

make_block_hash_to_score_lookup_key() (eth.db.schema.SchemaV1 static method), 63

make_block_number_to_hash_lookup_key() (eth.db.schema.SchemaV1 static method), 63

make_canonical_head_hash_lookup_key() (eth.db.schema.SchemaV1 static method), 63

make_receipt() (eth.abc.VirtualMachineAPI method), 47

make_receipt() (eth.vm.forks.byzantium.ByzantiumVM static method), 87

make_receipt() (eth.vm.forks.frontier.FrontierVM static method), 83

make_state_root() (eth.abc.AccountDatabaseAPI method), 41

make_state_root() (eth.abc.StateAPI method), 43

make_state_root() (eth.db.account.AccountDB method), 57

make_state_root() (eth.vm.state.BaseState method), 81

make_storage_root() (eth.abc.AccountStorageDatabaseAPI method), 40

make_storage_root() (eth.db.storage.AccountStorageDB method), 64

make_transaction_hash_to_block_lookup_key() (eth.db.schema.SchemaV1 static method), 63

Memory (class in eth.vm.memory), 76

memory_read() (eth.abc.ComputationAPI method), 38

memory_read() (eth.vm.computation.BaseComputation method), 74

memory_read_bytes() (eth.abc.ComputationAPI method), 38

memory_read_bytes() (eth.vm.computation.BaseComputation method), 74

memory_write() (eth.abc.ComputationAPI method), 38

memory_write() (eth.vm.computation.BaseComputation method), 74

MemoryAPI (class in eth.abc), 35

MemoryDB (class in eth.db.backends.memory), 58

Message (class in eth.vm.message), 76

MessageAPI (class in eth.abc), 34

mine_block() (eth.abc.MiningChainAPI method), 52

mine_block() (eth.abc.VirtualMachineAPI method), 47

mine_block() (eth.chains.base.MiningChain method), 56

mine_block() (eth.vm.base.VM method), 78

mine_block() (in module eth.tools.builder.chain), 69

mine_blocks() (in module eth.tools.builder.chain), 69

MiningChain (class in eth.chains.base), 55

MiningChainAPI (class in eth.abc), 52

MiningHeaderAPI (class in eth.abc), 29

N

name() (in module eth.tools.builder.chain), 69

new_trie() (eth.db.storage.StorageLookup method), 64

number (eth.abc.BlockAPI attribute), 31

O

Opcode (class in eth.vm.opcode), 76

OpcodeAPI (class in eth.abc), 34

origin (eth.abc.TransactionContextAPI attribute), 35

origin (eth.vm.transaction_context.BaseTransactionContext attribute), 82

OutOfBoundsRead, 65

OutOfGas, 65

output (eth.abc.ComputationAPI attribute), 39

output (eth.vm.computation.BaseComputation attribute), 74

override_transaction_context() (eth.abc.StateAPI method), 43

override_transaction_context() (eth.vm.state.BaseState method), 81

P

pack_block() (eth.abc.VirtualMachineAPI method), 47

pack_block() (eth.vm.base.VM method), 78

ParentNotFound, 65

peek() (eth.abc.CodeStreamAPI method), 36

peek() (eth.vm.code_stream.CodeStream method), 75

pending_items() (eth.db.diff.DBDiff method), 60

pending_keys() (eth.db.diff.DBDiff method), 60

persist() (eth.abc.AccountDatabaseAPI method), 41

persist() (eth.abc.AccountStorageDatabaseAPI method), 40

persist() (eth.abc.StateAPI method), 43

persist() (eth.db.account.AccountDB method), 57

persist() (eth.db.journal.JournalDB method), 62

persist() (eth.db.storage.AccountStorageDB method), 64

persist() (eth.vm.state.BaseState method), 81

persist_block() (eth.abc.ChainDatabaseAPI method), 34

persist_block() (eth.db.chain.ChainDB method), 60

persist_checkpoint_header() (eth.abc.HeaderDatabaseAPI method), 32

persist_checkpoint_header() (eth.db.header.HeaderDB method), 61

persist_header() (eth.abc.HeaderDatabaseAPI method), 32

persist_header() (eth.db.header.HeaderDB method), 62

persist_header_chain() (eth.abc.HeaderDatabaseAPI method), 32

persist_header_chain() (eth.db.header.HeaderDB method), 62

persist_trie_data_dict() (eth.abc.ChainDatabaseAPI method), 34

persist_trie_data_dict() (eth.db.chain.ChainDB method), 60

- persist_uncles() (eth.abc.ChainDatabaseAPI method), 34
 persist_uncles() (eth.db.chain.ChainDB method), 60
 PetersburgComputation (class in eth.vm.forks.istanbul.computation), 91
 PetersburgComputation (class in eth.vm.forks.petersburg.computation), 90
 PetersburgState (class in eth.vm.forks.petersburg.state), 90
 PetersburgVM (class in eth.vm.forks.petersburg), 89
 pop1_any() (eth.abc.StackAPI method), 35
 pop1_any() (eth.vm.stack.Stack method), 79
 pop1_bytes() (eth.abc.StackAPI method), 35
 pop1_bytes() (eth.vm.stack.Stack method), 79
 pop1_int() (eth.abc.StackAPI method), 35
 pop1_int() (eth.vm.stack.Stack method), 79
 pop_any() (eth.abc.StackAPI method), 35
 pop_any() (eth.vm.stack.Stack method), 79
 pop_bytes() (eth.abc.StackAPI method), 36
 pop_bytes() (eth.vm.stack.Stack method), 79
 pop_ints() (eth.abc.StackAPI method), 36
 pop_ints() (eth.vm.stack.Stack method), 80
 pre_state() (in module eth.tools.fixtures.fillers), 71
 precompiles (eth.abc.ComputationAPI attribute), 39
 precompiles (eth.vm.computation.BaseComputation attribute), 74
 prepare_child_message() (eth.abc.ComputationAPI method), 38
 prepare_child_message() (eth.vm.computation.BaseComputation method), 74
 prev_hashes (eth.abc.ExecutionContextAPI attribute), 37
 prev_hashes (eth.vm.execution_context.ExecutionContext attribute), 75
 previous_hashes (eth.abc.VirtualMachineAPI attribute), 48
 previous_hashes (eth.vm.base.VM attribute), 79
 push_bytes() (eth.abc.StackAPI method), 36
 push_bytes() (eth.vm.stack.Stack method), 80
 push_int() (eth.abc.StackAPI method), 36
 push_int() (eth.vm.stack.Stack method), 80
 PyEVMError, 65
- ## R
- raise_if_error() (eth.abc.ComputationAPI method), 38
 raise_if_error() (eth.vm.computation.BaseComputation method), 74
 read() (eth.abc.CodeStreamAPI method), 36
 read() (eth.abc.MemoryAPI method), 35
 read() (eth.vm.code_stream.CodeStream method), 75
 read() (eth.vm.memory.Memory method), 76
 read_bytes() (eth.abc.MemoryAPI method), 35
 read_bytes() (eth.vm.memory.Memory method), 76
 Receipt (class in eth.rlp.receipts), 67
 ReceiptAPI (class in eth.abc), 30
 ReceiptNotFound, 65
 record() (eth.abc.AccountDatabaseAPI method), 41
 record() (eth.abc.AccountStorageDatabaseAPI method), 40
 record() (eth.db.account.AccountDB method), 57
 record() (eth.db.storage.AccountStorageDB method), 64
 refund_gas() (eth.abc.ComputationAPI method), 38
 refund_gas() (eth.abc.GasMeterAPI method), 34
 refund_gas() (eth.vm.computation.BaseComputation method), 74
 refund_gas() (eth.vm.gas_meter.GasMeter method), 76
 register_account_for_deletion() (eth.abc.ComputationAPI method), 38
 register_account_for_deletion() (eth.vm.computation.BaseComputation method), 74
 reset() (eth.db.journal.JournalDB method), 63
 return_gas() (eth.abc.ComputationAPI method), 38
 return_gas() (eth.abc.GasMeterAPI method), 34
 return_gas() (eth.vm.computation.BaseComputation method), 74
 return_gas() (eth.vm.gas_meter.GasMeter method), 76
 Revert, 65
 revert() (eth.abc.StateAPI method), 43
 revert() (eth.vm.state.BaseState method), 81
 rollback_trie() (eth.db.storage.StorageLookup method), 64
- ## S
- SchemaV1 (class in eth.db.schema), 63
 seek() (eth.abc.CodeStreamAPI method), 36
 seek() (eth.vm.code_stream.CodeStream method), 75
 sender (eth.abc.SignedTransactionAPI attribute), 31
 set() (eth.abc.AccountStorageDatabaseAPI method), 40
 set() (eth.abc.DatabaseAPI method), 32
 set() (eth.db.backends.base.BaseDB method), 58
 set() (eth.db.storage.AccountStorageDB method), 64
 set_balance() (eth.abc.AccountDatabaseAPI method), 41
 set_balance() (eth.abc.StateAPI method), 43
 set_balance() (eth.db.account.AccountDB method), 57
 set_balance() (eth.vm.state.BaseState method), 81
 set_block_transactions() (eth.abc.VirtualMachineAPI method), 47
 set_block_transactions() (eth.vm.base.VM method), 78
 set_code() (eth.abc.AccountDatabaseAPI method), 41
 set_code() (eth.abc.StateAPI method), 43
 set_code() (eth.db.account.AccountDB method), 57
 set_code() (eth.vm.state.BaseState method), 81
 set_nonce() (eth.abc.AccountDatabaseAPI method), 41
 set_nonce() (eth.abc.StateAPI method), 44
 set_nonce() (eth.db.account.AccountDB method), 57
 set_nonce() (eth.vm.state.BaseState method), 81
 set_storage() (eth.abc.AccountDatabaseAPI method), 41
 set_storage() (eth.abc.StateAPI method), 44

- set_storage() (eth.db.account.AccountDB method), 57
 - set_storage() (eth.vm.state.BaseState method), 81
 - setup_main_filler() (in module eth.tools.fixtures.fillers.common), 71
 - should_burn_gas (eth.abc.ComputationAPI attribute), 39
 - should_burn_gas (eth.vm.computation.BaseComputation attribute), 74
 - should_erase_return_data (eth.abc.ComputationAPI attribute), 39
 - should_erase_return_data (eth.vm.computation.BaseComputation attribute), 75
 - should_return_gas (eth.abc.ComputationAPI attribute), 39
 - should_return_gas (eth.vm.computation.BaseComputation attribute), 75
 - SignedTransactionAPI (class in eth.abc), 31
 - snapshot() (eth.abc.StateAPI method), 44
 - snapshot() (eth.vm.state.BaseState method), 81
 - SpuriousDragonComputation (class in eth.vm.forks.spurious_dragon.computation), 87
 - SpuriousDragonState (class in eth.vm.forks.spurious_dragon.state), 86
 - SpuriousDragonVM (class in eth.vm.forks.spurious_dragon), 86
 - Stack (class in eth.vm.stack), 79
 - stack_dup() (eth.abc.ComputationAPI method), 39
 - stack_dup() (eth.vm.computation.BaseComputation method), 74
 - stack_pop1_any() (eth.abc.StackManipulationAPI method), 36
 - stack_pop1_bytes() (eth.abc.StackManipulationAPI method), 36
 - stack_pop1_int() (eth.abc.StackManipulationAPI method), 36
 - stack_pop_any() (eth.abc.StackManipulationAPI method), 36
 - stack_pop_bytes() (eth.abc.StackManipulationAPI method), 36
 - stack_pop_ints() (eth.abc.StackManipulationAPI method), 37
 - stack_push_bytes() (eth.abc.StackManipulationAPI method), 37
 - stack_push_int() (eth.abc.StackManipulationAPI method), 37
 - stack_swap() (eth.abc.ComputationAPI method), 39
 - stack_swap() (eth.vm.computation.BaseComputation method), 74
 - StackAPI (class in eth.abc), 35
 - StackDepthLimit, 65
 - StackManipulationAPI (class in eth.abc), 36
 - state (eth.abc.VirtualMachineAPI attribute), 48
 - state (eth.vm.base.VM attribute), 79
 - state_in_temp_block() (eth.abc.VirtualMachineAPI method), 47
 - state_in_temp_block() (eth.vm.base.VM method), 78
 - state_root (eth.abc.AccountDatabaseAPI attribute), 41
 - state_root (eth.abc.StateAPI attribute), 44
 - state_root (eth.db.account.AccountDB attribute), 57
 - state_root (eth.vm.state.BaseState attribute), 82
 - StateAPI (class in eth.abc), 42
 - StateRootNotFound, 65
 - StorageLookup (class in eth.db.storage), 64
 - swap() (eth.abc.StackAPI method), 36
 - swap() (eth.vm.stack.Stack method), 80
- ## T
- TangerineWhistleComputation (class in eth.vm.forks.tangerine_whistle.computation), 86
 - TangerineWhistleState (class in eth.vm.forks.tangerine_whistle.state), 86
 - TangerineWhistleVM (class in eth.vm.forks.tangerine_whistle), 85
 - timestamp (eth.abc.ExecutionContextAPI attribute), 37
 - timestamp (eth.abc.StateAPI attribute), 44
 - timestamp (eth.vm.execution_context.ExecutionContext attribute), 75
 - timestamp (eth.vm.state.BaseState attribute), 82
 - touch_account() (eth.abc.AccountDatabaseAPI method), 41
 - touch_account() (eth.abc.StateAPI method), 44
 - touch_account() (eth.db.account.AccountDB method), 57
 - touch_account() (eth.vm.state.BaseState method), 82
 - transaction_context_class (eth.vm.forks.frontier.state.FrontierState attribute), 84
 - transaction_executor_class (eth.vm.forks.frontier.state.FrontierState attribute), 84
 - transaction_executor_class (eth.vm.forks.spurious_dragon.state.SpuriousDragonState attribute), 87
 - TransactionContextAPI (class in eth.abc), 35
 - TransactionExecutorAPI (class in eth.abc), 41
 - TransactionFieldsAPI (class in eth.abc), 30
 - TransactionNotFound, 65
- ## U
- UnsignedTransactionAPI (class in eth.abc), 30
- ## V
- validate() (eth.abc.BaseTransactionAPI method), 30
 - validate() (eth.abc.SignedTransactionAPI method), 31
 - validate() (eth.rlp.transactions.BaseTransaction method), 67

[validate\(\)](#) (eth.rlp.transactions.BaseTransactionMethods method), [67](#) [WriteProtection](#), [65](#)
[validate_block\(\)](#) (eth.abc.ChainAPI method), [51](#)
[validate_block\(\)](#) (eth.abc.VirtualMachineAPI method), [47](#)
[validate_block\(\)](#) (eth.chains.base.Chain method), [55](#)
[validate_block\(\)](#) (eth.vm.base.VM method), [78](#)
[validate_chain\(\)](#) (eth.abc.ChainAPI method), [51](#)
[validate_chain\(\)](#) (eth.chains.base.BaseChain method), [52](#)
[validate_chain_extension\(\)](#) (eth.abc.ChainAPI method), [51](#)
[validate_chain_extension\(\)](#) (eth.chains.base.BaseChain method), [52](#)
[validate_gaslimit\(\)](#) (eth.abc.ChainAPI method), [52](#)
[validate_gaslimit\(\)](#) (eth.chains.base.Chain method), [55](#)
[validate_header\(\)](#) (eth.abc.VirtualMachineAPI class method), [47](#)
[validate_header\(\)](#) (eth.vm.base.VM class method), [79](#)
[validate_receipt\(\)](#) (eth.abc.ChainAPI method), [52](#)
[validate_receipt\(\)](#) (eth.abc.VirtualMachineAPI class method), [47](#)
[validate_receipt\(\)](#) (eth.chains.base.Chain method), [55](#)
[validate_receipt\(\)](#) (eth.vm.base.VM class method), [79](#)
[validate_receipt\(\)](#) (eth.vm.forks.byzantium.ByzantiumVM class method), [88](#)
[validate_seal\(\)](#) (eth.abc.ChainAPI method), [52](#)
[validate_seal\(\)](#) (eth.abc.VirtualMachineAPI method), [48](#)
[validate_seal\(\)](#) (eth.chains.base.Chain method), [55](#)
[validate_seal\(\)](#) (eth.vm.base.VM method), [79](#)
[validate_seal_extension\(\)](#) (eth.abc.VirtualMachineAPI method), [48](#)
[validate_seal_extension\(\)](#) (eth.vm.base.VM method), [79](#)
[validate_transaction\(\)](#) (eth.abc.StateAPI method), [44](#)
[validate_transaction\(\)](#) (eth.abc.TransactionExecutorAPI method), [42](#)
[validate_transaction\(\)](#) (eth.vm.forks.frontier.state.FrontierState method), [84](#)
[validate_transaction\(\)](#) (eth.vm.forks.homestead.state.HomesteadState method), [85](#)
[validate_transaction_against_header\(\)](#) (eth.abc.VirtualMachineAPI method), [48](#)
[validate_uncle\(\)](#) (eth.abc.VirtualMachineAPI class method), [48](#)
[validate_uncle\(\)](#) (eth.vm.base.VM class method), [79](#)
[validate_uncles\(\)](#) (eth.abc.ChainAPI method), [52](#)
[validate_uncles\(\)](#) (eth.chains.base.Chain method), [55](#)
[VirtualMachineAPI](#) (class in eth.abc), [44](#)
[VM](#) (class in eth.vm.base), [76](#)
[VMError](#), [65](#)
[VMNotFound](#), [65](#)

W

[write\(\)](#) (eth.abc.MemoryAPI method), [35](#)
[write\(\)](#) (eth.vm.memory.Memory method), [76](#)