
py-evm Documentation

Release 0.10.1-beta.1

Ethereum Foundation

Apr 19, 2024

GENERAL

| | | |
|----------|----------------------------|------------|
| 1 | Goals | 3 |
| 2 | Usage | 5 |
| 3 | Further reading | 7 |
| 4 | Table of contents | 9 |
| | Python Module Index | 179 |
| | Index | 181 |

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.

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

USAGE

Check out our [guides](#) to get started using the `py-evm` library.

FURTHER READING

Here are a couple more useful links to check out.

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

TABLE OF CONTENTS

4.1 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.

4.1.1 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

4.1.2 Usage

Check out our [guides](#) to get started using the `py-evm` library.

4.1.3 Further reading

Here are a couple more useful links to check out.

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

4.2 Release Notes

4.2.1 py-evm v0.10.1-beta.1 (2024-04-18)

Bugfixes

- Remove `scripts/__init__.py` so that `scripts` doesn't get imported as a package. Also removes the `scripts/` directory from the wheel. (#2172)

Features

- Add support for python 3.12. (#2171)

Internal Changes - For py-evm Contributors

- Exclude *scripts* when building the wheel. (#2170)
- Update link to the getting started guide in the README. (#2173)
- Remove `scripts/` directory on doc build (#2174)

4.2.2 py-evm v0.10.0-beta.6 (2024-04-05)

Bugfixes

- Fix `__getattr__()` for `KeyMapDB` and thus allow it to be copied / deep copied. (#2116)
- `integer.to_bytes()` requires size and byteorder below py311 and our fixture tests only use py311. Compare the first byte of versioned hashes by indexing instead. (#2168)

4.2.3 py-evm v0.10.0-beta.5 (2024-04-05)

Bugfixes

- Use the current VM's header class to check valid fields for `vm.pack_block()`. (#2165)
- Properly configure the `CancunBlock` class to use the `CancunReceiptBuilder`. (#2166)

4.2.4 py-evm v0.10.0-beta.4 (2024-03-18)

Bugfixes

- Clear existing transient storage db instead of resetting and creating a new one (#2159)

4.2.5 py-evm v0.10.0-beta.3 (2024-03-18)

Bugfixes

- bugfix: Ensure a `type_id` for `SpoofTransaction` when unsigned -> signed spoofing. This defaults to `None` for legacy and uses the `_type_id` for unsigned typed txns. (#2157)

Internal Changes - For py-evm Contributors

- Use some general state tests for transaction tests since they have similar formats. This yielded a decent amount of new transaction tests. (#2157)

4.2.6 py-evm v0.10.0-beta.2 (2024-03-15)

Bugfixes

- bugfix: Address issues instantiating VM at Cancun transition. (#2156)

4.2.7 py-evm v0.10.0-beta.1 (2024-03-15)

Breaking Changes

- Remove `memory_read` from `ComputationAPI` interface and `Computation` implementation. Use `memory_read_bytes` in its place for call data read. (#2140)

Features

- Implement EIP-4788: Add `parent_beacon_block_root` to execution block headers. (#2135)
- Implement EIP-1153: Transient Storage. (#2142)
- Implement EIP-6780: Self-destruct only in same transaction. (#2148)
- Implement EIP-4844 and EIP-7516: Blob transactions, `BLOBHASH` opcode, `BLOBBASEFEE` opcode. (#2151)

Internal Changes - For py-evm Contributors

- Drop the concept of a mining header, post-merge. (#2134)
- Fix epub docs build issue, add pdf and epub docs builds to CI (#2137)
- Update *ethereum/tests* submodule to version v13.1. (#2149)

4.2.8 py-evm v0.9.0-beta.1 (2024-02-05)

Breaking Changes

- Drop python 3.7 support (#2128)

Bugfixes

- Accept `type==0` as legacy a transaction. (#2136)

Internal Changes - For py-evm Contributors

- Merge updates from the project template, including using `pre-commit` for linting and changing the name of the `master` branch to `main` (#2128)
- Update *ethereum/tests* test fixture to use v13. (#2136)

Performance Improvements

- Performance improvements; code refactor; some cleanup. (#2076)

4.2.9 py-evm v0.8.0-beta.1 (2023-10-09)

Features

- Python 3.10 and 3.11 support. (#2088)

Breaking changes

- Remove dependency on `pyethash`, `pysha3`, and `pycryptodome` packages and internalize the ethash algorithm implementation into Python code with significant loss of performance, in an effort to un-prioritize proof-of-work consensus and logic. (#2121)

4.2.10 py-evm v0.7.0-alpha.4 (2023-07-24)

Bugfixes

- `eth_now` now returns a utc timestamp instead of a local timestamp (#2119)

Internal Changes - For py-evm Contributors

- Bumped mypy version to 1.4.0 (#2117)

4.2.11 py-evm v0.7.0-alpha.3 (2023-06-08)

Bugfixes

- Updated *CodeStream* slot name *pc* to *program_counter* to match the attribute name (#2109)
- Bring CREATE and CREATE2 logic up to speed wrt changes to EIP-2681 (high nonce). (#2110)

Internal Changes - For py-evm Contributors

- Update *fixtures* (ethereum/tests) version to v12.2 and turn on all Shanghai fork tests since EOF is no longer in Shanghai. (#2108)
- Fix some failing tests by properly decoding the tx bytes provided by the Transaction test fixtures. (#2111)
- bump version for flake8, flake8-bugbear, and mypy, and cleanup *tox.ini* (#2113)

4.2.12 py-evm v0.7.0-alpha.2 (2023-05-11)

Bugfixes

- Add missing receipt builder for the *ShanghaiBlock* class. (#2105)

Internal Changes - For py-evm Contributors

- Added `[isort](https://pycqa.github.io/isort/)` for automatically sorting python imports. (#2094)
- pull in less-sensitive updates from the python project template (#2095)
- Update `pip` version sitting in the circleci image before installing and running `tox`. Install `tox` at the sys level to help avoid `virtualenv` version conflicts. (#2102)
- Refactored the computation class hierarchy and cleaned up the code along the way. Some abstract API classes have more of the underlying properties that the subclasses implement. (#2106)
- added *black* to lint dependencies and `isort`ed scripts directory`` (#2107)

Miscellaneous changes

- [#2083](#)

4.2.13 py-evm 0.7.0-alpha.1 (2023-04-10)

Features

- Add Shanghai hard fork support. ([#2093](#))

Breaking changes

- `configure_header()` now accepts a difficulty function as a `kwarg` rather than positional `arg` due to POS priority. ([#2093](#))

4.2.14 py-evm 0.6.1-alpha.2 (2022-12-16)

Miscellaneous internal changes

- [#2090](#)

4.2.15 py-evm 0.6.1-alpha.1 (2022-11-14)

Features

- Support for the paris fork a.k.a. “the merge”. ([#2080](#))

Bugfixes

- Use the `DIFFICULTY_MINIMUM` more appropriately as the lower limit in all difficulty calculations. ([#2084](#))

Internal Changes - for Contributors

- Update towncrier version to remove double headers. ([#2077](#))
- Update openssl config on circleci builds to re-introduce `ripemd160` function by default. ([#2087](#))

Miscellaneous internal changes

- [#2078](#), [#2082](#), [#2085](#)

4.2.16 py-evm 0.6.0-alpha.1 (2022-08-22)

Features

- Gray glacier support without Merge transition since Merge is not yet supported ([#2072](#))

Bugfixes

- Arrow Glacier header serialization fixed to properly inherit from LondonBlockHeader ([#2047](#))

Deprecations and Removals

- Upgrade dependencies: eth-keys, eth-typing, eth-utils, py-ecc, rlp, trie ([#2068](#))
- Drop python 3.6 support ([#2070](#))

4.2.17 py-evm 0.5.0-alpha.3 (2022-01-26)

Bugfixes

- Downgrade upstream dependencies to allow only non-breaking changes. Once we're ready to cut web3.py v6 branch, we can pull in breaking changes from upstream dependencies. Namely, dropping Python 3.5 and 3.6. ([#2050](#))

4.2.18 py-evm 0.5.0-alpha.2 (2021-12-16)

Features

- Arrow Glacier Support
 - Implement [EIP-4345](#) for Arrow Glacier support. ([#2045](#))

Miscellaneous internal changes

- [#2040](#), [#2045](#), [#2048](#)

4.2.19 py-evm 0.5.0-alpha.1 (2021-10-13)

Features

- ([#2038](#))
 - Add `validate()` method and `intrinsic_gas` property to `UnsignedAccessListTransaction`
 - Add `validate()` method and `intrinsic_gas` property to `UnsignedDynamicFeeTransaction`

Improved Documentation

- Updated the reference to the project template in the docs to <https://github.com/ethereum/ethereum-python-project-template> and changed the location in the git clone command accordingly. (#2032)
- Documentation updates to use latest py-evm version, grammar updates, python version updates, replace Gitter link with Discord link, and change [.dev] -> “[.dev]” in docs for better compatibility across shells (#2036)

4.2.20 py-evm 0.5.0-alpha.0 (2021-09-30)

Features

London Support

- Pass all London tests from the ethereum/tests repo (#2017)
- Implement EIP-1559 for London support. (#2013)
- Implement EIP-3198 for London support. (#2015)
- Implement EIP-3554 for London support. (#2018)
- Implement EIP-3541 for London support. (#2018)
- Implement EIP-3529 for London support. (#2020)

Bugfixes

- Replace local timestamps with UTC timestamps (#2013)
 - Use UTC timestamp instead of local time zone, when creating a header.
 - Use UTC for clique validation.
- Was overly permissive on the header gas limit by one gas. (#2021)
 - Make header gas limit more restrictive by one, in various places.
 - Validate uncle gas limits are within bounds of parent. This was previously not validated at all.
- Erase return data for exceptions with *erases_return_data* flag set to True and for CREATE / CREATE2 computations with insufficient funds (#2023)

Deprecations and Removals

- Removed old options and methods for creating a header, now that headers vary by fork. (#2013)
 - `eth.rlp.headers.BlockHeader.from_parent()` is gone, because you should always use the VM to create a header (to make sure you get the correct type).
 - Can no longer supply some fields to the genesis, like bloom and parent_hash.

Internal Changes - for Contributors

- Misc test improvements (#2013)
 - some test_vm fixes:
 - * use the correctly paired VMs in PoW test
 - * make sure *only* the block number is invalid in block number validity test
 - more robust test fixture name generation
 - run a newer version of the lint test from *make lint*
- Various upgrades and related updates (#2016)
 - Upgrade pytest and pytest-xdist. Caching was causing very slow test runs locally in pytest v5
 - Update ethereum/tests (Updated in various PRs. At release time, checked out at v10.0)
 - Remove json-fixture caching to resolve stale cache issues that caused test failures (at some expense to speed)
 - Make xdist failures show up correctly in the transition tests
- During fixture tests, verify that the generated genesis block matches the fixture's RLP-encoding. (#2022)
- Squash sphinx warnings with a small documentation reorg. (#2021)

4.2.21 py-evm 0.4.0-alpha.4 (2021-04-07)

Features

- Add Python 3.9 support (#1999)

Internal Changes - for Contributors

- Update ethereum/tests fixture to v8.0.2, mark some new tests as too slow for CI. (#1998)

Miscellaneous internal changes

- Update blake2b-py requirement from $\geq 0.1.2$ to $\geq 0.1.4$ (#1999)

4.2.22 py-evm 0.4.0-alpha.3 (2021-03-24)

Features

- Expose a `type_id` on all transactions. It is `None` for legacy transactions. (#1996)
- Add new `LegacyTransactionFieldsAPI`, with a `v` field for callers that want to access `v` directly. (#1997)

Bugfixes

- Fix a crash in `eth.chains.base.Chain.get_transaction_receipt()` and `eth.chains.base.Chain.get_transaction_receipt_by_index()` that resulted in this exception: `TypeError: get_receipt_by_index() got an unexpected keyword argument 'receipt_builder'` (#1994)

4.2.23 py-evm 0.4.0-alpha.2 (2021-03-22)

Bugfixes

- Add Berlin block numbers for Goerli and Ropsten. Correct the type signature for `TransactionBuilderAPI` and `ReceiptBuilderAPI`, because `deserialize()` can take a list of bytes for the legacy types. (#1993)

4.2.24 py-evm 0.4.0-alpha.1 (2021-03-22)

Features

- Berlin Support
 - EIP-2718: Typed Transactions – no new functionality, really. It is mostly refactoring in preparation for EIP-2930. (which does churn the code a fair bit) (#1973)
 - EIP-2930: Optional access lists. Implement the new transaction type 1, which pre-warms account & storage caches from EIP-2929, and adds first-class `chain_id` support. (#1975)
 - EIP-2929: Gas cost increases for state access opcodes. Charge more for cold-cache access of account and storage. (#1974)
 - EIP-2565: Update `ModExp` precompile gas cost calculation (#1976 & #1989)

Bugfixes

- Uncles with the same timestamp as their parents are invalid. Reject them, and add the test from `ethereum/tests`. (#1979)

Performance improvements

- Got a >10x speedup of some benchmarks and other tests, by adding a new `eth.chains.base.MiningChain.mine_all()` API and using it. This is a public API, and should be used whenever all the transactions are known up front, to get a significant speedup. (#1967)

Internal Changes - for Contributors

- Upgrade tests fixtures to v8.0.1, with Berlin tests. Skipped several slow tests in Istanbul. Added `pytest-timeout` to limit annoyance of new slow tests. (#1971, #1987, #1991, #1989)
- Make sure Berlin is tested across all core tests. (also patched in some missing Muir Glacier ones) (#1977)

4.2.25 py-evm 0.3.0-alpha.20 (2020-10-21)

Bugfixes

- Upgrade rlp library to v2.0.0 stable, which is friendlier to 32-bit and other architectures. Downstream applications can choose to explicitly install the rust implementation with `pip install rlp[rust-backend]`. ([d553bd](#))

4.2.26 py-evm 0.3.0-alpha.19 (2020-08-31)

Features

- Add a new hook `eth.abc.VirtualMachineAPI.transaction_applied_hook()` which is triggered after each transaction in `apply_all_transactions`, which is called by `import_block`. The first use case is reporting progress in the middle of Beam Sync. ([#1950](#))

Performance improvements

- Upgrade rlp library to v2.0.0-a1 which uses faster rust based encoding/decoding. ([#1951](#))

Deprecations and Removals

- Removed unused and broken `add_uncle` API on `FrontierBlock` and consequentially on all other derived block classes. ([#1949](#))

Internal Changes - for Contributors

- Improve type safety by ensuring `abc` types do not inherit from `rlp.Serializable` which implicitly has type `Any`. ([#1948](#))

Miscellaneous internal changes

- [#1953](#)

4.2.27 py-evm 0.3.0-alpha.18 (2020-06-25)

Features

- Expose `get_chain_gaps()` on `ChainDB` to track gaps in the chain of blocks. ([#1947](#))

Internal Changes - for Contributors

- Allow *mine_block* of chain builder tools to take a `transactions` parameter. This makes it easier to model test scenarios that depend on creating blocks with transactions. (#1947)
- upgrade to Upgrade py-trie to the new v2.0.0-alpha.2 with fixed `TraversedPartialPath`

4.2.28 py-evm 0.3.0-alpha.17 (2020-06-02)

Features

- Added support for Python 3.8. (#1940)
- Methods now raise *BlockNotFound* when retrieving a block, and some part of the block is missing. These methods used to raise a `KeyError` if transactions were missing, or a `HeaderNotFound` if uncles were missing:
 - `eth.db.chain.ChainDB.get_block_by_header()`
 - `eth.db.chain.ChainDB.get_block_by_hash()` (it still raises a `HeaderNotFound` if there is no header matching the given hash)
 - *Block.from_header()* (#1943)

Bugfixes

- A number of fixes related to checkpoints and persisting old headers, especially when we try to persist headers that don't match the checkpoints.
 - A new exception *CheckpointsMustBeCanonical* raised when persisting a header that is not linked to a previously-saved checkpoint. (note: we now explicitly save checkpoints)
 - More broadly, any block persist that would cause the checkpoint to be decanonicalized will raise the *CheckpointsMustBeCanonical*.
 - Re-insert gaps in the chain when a checkpoint and (parent or child) header do not link
 - De-canonicalize all children of orphans. (Previously, only decanonicalized headers with block numbers that matched the new canonical headers)
 - Added some new hypothesis tests to get more confidence that we covered most cases
 - When filling a gap, if there's an existing child that is not a checkpoint and doesn't link to the parent, then the parent block wins, and the child block is de-canonicalized (and gap added). (#1929)

Internal Changes - for Contributors

- Upgrade py-trie to the new v2.0.0-alpha.1, and pin it for stability. (#1935)
- Improve the error when transaction nonce is invalid: include expected and actual. (#1936)

4.2.29 py-evm 0.3.0-alpha.16 (2020-05-27)

Features

- Expose `get_header_chain_gaps()` API on HeaderDB to track chain gaps (#1924)
- Add a new `persist_unexecuted_block` API to ChainDB. This API should be used to persist a block without executing the EVM on it. The API is used by syncing strategies that do not execute all blocks but fill old blocks back in (e.g. `beam` or `fast sync`) (#1925)
- Update the allowable version of `py_ecc` library. (#1934)

4.2.30 py-evm 0.3.0-alpha.15 (2020-04-14)

Features

- `eth.chains.base.Chain.import_block()` now returns some meta-information about the witness. You can get a list of trie node hashes needed to build the witness, as well as the accesses of accounts, storage slots, and bytecodes. (#1917)

Internal Changes - for Contributors

- Use a more recent `eth-keys`, which calls an `eth-typing` that's not deprecated. (#1665)
- Upgrade `pytest-xdist` from 1.18.1 to 1.31.0, to fix a CI crash. (#1917)
- Added `KeyAccessLoggerDB` and its atomic twin; faster `make validate-docs` (but you have to remember to `pip install -e .[doc]` yourself); `str(block)` now includes some bytes of the block hash. (#1918)
- Fix for creating a duplicate “ghost” `Computation` that was never used. It didn't break anything, but was inelegant and surprising to get extra objects created that were mostly useless. This was achieved by changing `eth.abc.ComputationAPI.apply_message()` and `eth.abc.ComputationAPI.apply_create_message()` to be class methods. (#1921)

4.2.31 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)

4.2.32 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)

4.2.33 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)

4.2.34 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)

4.2.35 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)

4.2.36 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)

4.2.37 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)

4.2.38 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)

4.2.39 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>=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)

4.2.40 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)

4.2.41 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)

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

Bugfixes

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

4.2.43 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)

4.2.44 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

4.2.45 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

4.2.46 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

4.2.47 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

4.3 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.

4.3.1 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)
```

4.3.2 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
```

(continues on next page)

(continued from previous page)

```
>>> # Giving funds to some address
>>> SOME_ADDRESS = b'\x85\x82\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 = {
...     'difficulty': constants.GENESIS_DIFFICULTY,
... }

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

4.3.3 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)
1000000000000000000
```

4.3.4 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.

4.4 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](#).

4.4.1 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 Python on Ubuntu

Py-EVM requires Python 3 as well as some tools to compile its dependencies. On Ubuntu, the `python3.9-dev` package contains everything we need. Run the following command to install it.

```
apt-get install python3.9-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
```

Installing Python on macOS

First, install Python 3 with brew:

```
brew install python3
```

Installing py-evm

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, make sure to have the latest version of `pip` so that all dependencies can be installed correctly:

```
pip3 install -U pip
```

Finally, 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

4.4.2 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/ethereum/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-evm==0.5.0a0",
],
```

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 = {
...     'difficulty': constants.GENESIS_DIFFICULTY,
... }

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

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

>>> mock_address_balance = chain.get_vm().state.get_balance(MOCK_ADDRESS)

>>> print(f"The balance of address {encode_hex(MOCK_ADDRESS)} is {mock_address_balance} wei")
The balance of address 0x0000000000000000000000000000000000000000 is 100000000000000000000000000000000 wei
```

Running 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
↳10000000000000000000000000000000000000000000000000000000000000000 wei
```

4.4.3 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`

4.4.4 Understanding the mining process

Note: Proof-of-Work (PoW) mining is no longer used for achieving consensus on Ethereum. Newer virtual machines, beginning with the ParisVM, assume a Proof-of-Stake (PoS) consensus mechanism which lies beyond the scope of the execution layer. This guide is for educational purposes only.

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 that was once important for achieving consensus on mainnet Ethereum.

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 is 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(
        get_dataset_full_size(block_number),
        cache,
        mining_hash,
        nonce,
    )
    if mining_output["mix_digest"] != mix_hash:
        raise ValidationError(
            f"mix hash mismatch; expected: {encode_hex(mining_output['mix_digest'])} "
            f"!= actual: {encode_hex(mix_hash)}.\\n      "
            f"Mix hash calculated from block #{block_number},\\n      "
            f"mine hash: {encode_hex(mining_hash)},\\n      "
            f"nonce: {encode_hex(nonce)},\\n      "
            f"difficulty: {difficulty}"
        )
    result = big_endian_to_int(mining_output["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 = {
    'difficulty': 1,
    'gas_limit': 3141592,
    'timestamp': 1514764800,
}
```

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),
```

(continues on next page)

(continued from previous page)

```

    ))
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_result = chain.get_vm().finalize_block(chain.get_block())
block = block_result.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.

```
from eth_keys import keys
from eth_utils import decode_hex
from eth_typing import Address

SENDER_PRIVATE_KEY = keys.PrivateKey(
    decode_hex('0x45a915e4d060149eb4365960e6a7a45f334393093061116b197e3240065ff2d8')
)

SENDER = Address(SENDER_PRIVATE_KEY.public_key.to_canonical_address())

RECEIVER = Address(b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02')
```

One thing that strikes out here is that we only need the plain address for the receiver whereas for the sender we are obtaining an address derived from the `SENDER_PRIVATE_KEY`. That's because we obviously can not send transactions from an address that we don't have the private key to sign it for.

With sender and receiver prepared, let's create the actual transaction.

```
vm = chain.get_vm()
nonce = vm.state.get_nonce(SENDER)

tx = vm.create_unsigned_transaction(
    nonce=nonce,
    gas_price=0,
    gas=1000000,
    to=RECEIVER,
    value=0,
    data=b'',
)
```

Every transaction needs a **nonce** not to be confused with the nonce that we previously mined as part of the PoW algorithm. The *transaction nonce* serves as a counter to ensure all transactions from one address are processed in order. We retrieve the current nonce by calling `get_nonce(sender)()`.

Once we have the nonce we can call `create_unsigned_transaction()` and pass the nonce among the rest of the transaction attributes as key-value pairs.

- **nonce** - Number of transactions sent by the sender
- **gas_price** - Number of Wei to pay per unit of gas
- **gas** - Maximum amount of gas the transaction is allowed to consume before it gets rejected
- **to** - Address of transaction recipient
- **value** - Number of Wei to be transferred to the recipient

The last step we need to do before we can add the transaction to a block is to sign it with the private key which is as simple as calling `as_signed_transaction()` with the `SENDER_PRIVATE_KEY`.

```
signed_tx = tx.as_signed_transaction(SENDER_PRIVATE_KEY)
```

Finally, we can call `apply_transaction()` and pass along the `signed_tx`.

```
chain.apply_transaction(signed_tx)
```

What follows is the complete script that demonstrates how to mine a single block with one simple zero value transfer transaction.

```
>>> from eth_keys import keys
>>> from eth_utils import decode_hex
>>> from eth_typing import Address
>>> from eth import constants
>>> from eth.chains.base import MiningChain
>>> from eth.consensus.pow import mine_pow_nonce
>>> from eth.vm.forks.byzantium import ByzantiumVM
>>> from eth.db.atomic import AtomicDB

>>> GENESIS_PARAMS = {
...     'difficulty': 1,
...     'gas_limit': 3141592,
...     # We set the timestamp, just to make this documented example reproducible.
...     # In common usage, we remove the field to let py-evm choose a reasonable default.
...     'timestamp': 1514764800,
... }

>>> SENDER_PRIVATE_KEY = keys.PrivateKey(
...     decode_hex('0x45a915e4d060149eb4365960e6a7a45f334393093061116b197e3240065ff2d8')
... )

>>> SENDER = Address(SENDER_PRIVATE_KEY.public_key.to_canonical_address())

>>> RECEIVER = Address(b'\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\x02')

>>> klass = MiningChain.configure(
...     __name__='TestChain',
...     vm_configuration=(
...         (constants.GENESIS_BLOCK_NUMBER, ByzantiumVM),
...     ))

>>> chain = klass.from_genesis(AtomicDB(), GENESIS_PARAMS)
>>> genesis = chain.get_canonical_block_header_by_number(0)
>>> vm = chain.get_vm()

>>> nonce = vm.state.get_nonce(SENDER)

>>> tx = vm.create_unsigned_transaction(
...     nonce=nonce,
...     gas_price=0,
...     gas=1000000,
...     to=RECEIVER,
...     value=0,
...     data=b'',
... )

>>> signed_tx = tx.as_signed_transaction(SENDER_PRIVATE_KEY)
```

(continues on next page)

(continued from previous page)

```
>>> chain.apply_transaction(signed_tx)
(<ByzantiumBlock(#Block #1...)

>>> # Normally, we can let the timestamp be chosen automatically, but
>>> # for the sake of reproducing exactly the same block every time,
>>> # we will set it manually here:
>>> chain.set_header_timestamp(genesis.timestamp + 1)

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

>>> chain.mine_block(mix_hash=mix_hash, nonce=nonce)
<ByzantiumBlock(#Block #1-0xe372..385c)>
```

4.4.5 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.

4.5 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.

4.5.1 ABC

Abstract base classes for documented interfaces

MiningHeaderAPI

class `eth.abc.MiningHeaderAPI`

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.

abstract `as_dict()` → `Dict[Hashable, Any]`

Return a dictionary representation of the header.

abstract `build_changeset(*args: Any, **kwargs: Any) → Any`

Open a changeset to modify the header.

abstract `property base_fee_per_gas: int | None`

Return the base fee per gas of the block.

Set to None in pre-EIP-1559 (London) header.

block_number: `BlockNumber`

bloom: `int`

coinbase: `Address`

difficulty: `int`

extra_data: `bytes`

gas_limit: `int`

gas_used: `int`

abstract `property hash: Hash32`

Return the hash of the block header.

abstract `property hex_hash: str`

Return the hash as a hex string.

abstract `property is_genesis: bool`

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

abstract `property mining_hash: Hash32`

Return the mining hash of the block header.

parent_hash: Hash32

receipt_root: Hash32

state_root: Hash32

timestamp: `int`

transaction_root: Hash32

uncles_hash: Hash32

abstract property withdrawals_root: Hash32 | `None`

Return the withdrawals root of the block.

Set to None in pre-Shanghai header.

BlockHeaderAPI

class `eth.abc.BlockHeaderAPI`

A class derived from [*MiningHeaderAPI*](#) to define a block header after it is sealed.

abstract `copy(*args: Any, **kwargs: Any) → BlockHeaderAPI`

Return a copy of the header, optionally overwriting any of its properties.

abstract property blob_gas_used: `int`

Return blob gas used.

abstract property excess_blob_gas: `int`

Return excess blob gas.

mix_hash: Hash32

nonce: `bytes`

abstract property parent_beacon_block_root: Hash32 | `None`

Return the hash of the parent beacon block.

LogAPI

class `eth.abc.LogAPI`

A class to define a written log.

address: Address

abstract property bloomables: `Tuple[bytes, ...]`

data: `bytes`

topics: `Sequence[int]`

ReceiptAPI

class eth.abc.ReceiptAPI

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

copy(*args: *Any*, **kwargs: *Any*) → *ReceiptAPI*

Return a copy of the receipt, optionally overwriting any of its properties.

abstract encode() → *bytes*

This encodes a receipt, no matter if it's: a legacy receipt, a typed receipt, or the payload of a typed receipt.
See more context in decode.

abstract property bloom: *int*

abstract property bloom_filter: *BloomFilter*

abstract property gas_used: *int*

abstract property logs: *Sequence[LogAPI]*

abstract property state_root: *bytes*

BaseTransactionAPI

class eth.abc.BaseTransactionAPI

A class to define all common methods of a transaction.

abstract copy(**overrides: *Any*) → *T*

Return a copy of the transaction.

abstract gas_used_by(computation: *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.

abstract 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.

abstract validate() → *None*

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

abstract property access_list: *Sequence[Tuple[Address, Sequence[int]]]*

Get addresses to be accessed by a transaction, and their storage slots.

abstract property intrinsic_gas: *int*

Convenience property for the return value of *get_intrinsic_gas*

TransactionFieldsAPI

class eth.abc.TransactionFieldsAPI

A class to define all common transaction fields.

abstract property blob_versioned_hashes: Sequence[Hash32]

abstract property chain_id: int | None

abstract property data: bytes

abstract property gas: int

abstract property gas_price: int

Will raise `AttributeError` if get or set on a 1559 transaction.

abstract property hash: Hash32

Return the hash of the transaction.

abstract property max_fee_per_blob_gas: int

abstract property max_fee_per_gas: int

Will default to gas_price if this is a pre-1559 transaction.

abstract property max_priority_fee_per_gas: int

Will default to gas_price if this is a pre-1559 transaction.

abstract property nonce: int

abstract property r: int

abstract property s: int

abstract property to: Address

abstract property value: int

UnsignedTransactionAPI

class eth.abc.UnsignedTransactionAPI

A class representing a transaction before it is signed.

abstract as_signed_transaction(*private_key: PrivateKey, chain_id: int | None = None*) →
SignedTransactionAPI

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

data: bytes

gas: int

gas_price: int

nonce: int

to: Address

value: int

SignedTransactionAPI

class `eth.abc.SignedTransactionAPI(*args: Any, **kwargs: Any)`

as_dict() → `Dict[Hashable, Any]`

Return a dictionary representation of the transaction.

abstract check_signature_validity() → `None`

Check if the signature is valid. Raise a `ValidationError` if the signature is invalid.

abstract encode() → `bytes`

This encodes a transaction, no matter if it's: a legacy transaction, a typed transaction, or the payload of a typed transaction. See more context in `decode`.

abstract get_message_for_signing() → `bytes`

Return the bytestring that should be signed in order to create a signed transaction.

abstract get_sender() → `Address`

Get the 20-byte address which sent this transaction.

This can be a slow operation. `transaction.sender` is always preferred.

abstract make_receipt(*status: bytes, gas_used: int, log_entries: Tuple[Tuple[bytes, Tuple[int, ...], bytes], ...]*) → `ReceiptAPI`

Build a receipt for this transaction.

Transactions have this responsibility because there are different types of transactions, which have different types of receipts. (See access-list transactions, which change the receipt encoding)

Parameters

- **status** – success or failure (used to be the state root after execution)
- **gas_used** – cumulative usage of this transaction and the previous ones in the header
- **log_entries** – logs generated during execution

abstract validate() → `None`

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

abstract property is_signature_valid: `bool`

Return True if the signature is valid, otherwise False.

abstract property sender: `Address`

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

type_id: `int | None`

The type of EIP-2718 transaction

Each EIP-2718 transaction includes a type id (which is the leading byte, as encoded).

If this transaction is a legacy transaction, that it has no type. Then, `type_id` will be `None`.

abstract property y_parity: `int`

The bit used to disambiguate elliptic curve signatures.

The only values this method will return are 0 or 1.

BlockAPI

```
class eth.abc.BlockAPI(header: BlockHeaderAPI, transactions: Sequence[SignedTransactionAPI], uncles:
                        Sequence[BlockHeaderAPI], withdrawals: Sequence[WithdrawalAPI] | None = None)
```

A class to define a block.

```
copy(*args: Any, **kwargs: Any) → BlockAPI
```

Return a copy of the block, optionally overwriting any of its properties.

```
abstract classmethod from_header(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI) →
                                BlockAPI
```

Instantiate a block from the given header and the chaindb.

```
abstract classmethod get_receipt_builder() → Type[ReceiptBuilderAPI]
```

Return the receipt builder for the block.

```
abstract get_receipts(chaindb: ChainDatabaseAPI) → Tuple[ReceiptAPI, ...]
```

Fetch the receipts for this block from the given chaindb.

```
abstract classmethod get_transaction_builder() → Type[TransactionBuilderAPI]
```

Return the transaction builder for the block.

```
abstract property hash: Hash32
```

Return the hash of the block.

```
header: BlockHeaderAPI
```

```
abstract property is_genesis: bool
```

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

```
abstract property number: BlockNumber
```

Return the number of the block.

```
receipt_builder: Type[ReceiptBuilderAPI] = None
```

```
transaction_builder: Type[TransactionBuilderAPI] = None
```

```
transactions: Tuple[SignedTransactionAPI, ...]
```

```
uncles: Tuple[BlockHeaderAPI, ...]
```

```
withdrawals: Tuple[WithdrawalAPI, ...]
```

DatabaseAPI

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

A class representing a database.

```
abstract delete(key: bytes) → None
```

Delete the given key from the database.

```
abstract exists(key: bytes) → bool
```

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

```
abstract set(key: bytes, value: bytes) → None
```

Assign the value to the key.

AtomicDatabaseAPI

class eth.abc.AtomicDatabaseAPI(*args, **kwargs)

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

abstract `atomic_batch()` → AbstractContextManager[AtomicWriteBatchAPI]

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

HeaderDatabaseAPI

class eth.abc.HeaderDatabaseAPI(db: AtomicDatabaseAPI)

A class representing a database for block headers.

abstract `get_block_header_by_hash(block_hash: Hash32)` → BlockHeaderAPI

Return the block header for the given `block_hash`. Raise `HeaderNotFound` if no header with the given `block_hash` exists in the database.

abstract `get_canonical_block_hash(block_number: BlockNumber)` → Hash32

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.

abstract `get_canonical_block_header_by_number(block_number: BlockNumber)` → 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.

abstract `get_canonical_head()` → BlockHeaderAPI

Return the current block header at the head of the chain.

abstract `get_header_chain_gaps()` → Tuple[Tuple[Tuple[BlockNumber, BlockNumber], ...], BlockNumber]

Return information about gaps in the chain of headers. This consists of an ordered sequence of block ranges describing the integrity of the chain. Each block range describes a missing segment in the chain and each range is defined with inclusive boundaries, meaning the first value describes the first missing block of that segment and the second value describes the last missing block of the segment.

In addition to the sequences of block ranges a block number is included that indicates the number of the first header that is known to be missing at the very tip of the chain.

abstract `get_score(block_hash: Hash32)` → int

Return the score for the given `block_hash`.

abstract `header_exists(block_hash: Hash32)` → bool

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

abstract `persist_checkpoint_header(header: BlockHeaderAPI, score: int)` → None

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

abstract `persist_header(header: BlockHeaderAPI)` → Tuple[Tuple[BlockHeaderAPI, ...], Tuple[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

```
abstract persist_header_chain(headers: Sequence[BlockHeaderAPI], genesis_parent_hash: Hash32 |
                                None = None) → Tuple[Tuple[BlockHeaderAPI, ...],
                                Tuple[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.

db: [AtomicDatabaseAPI](#)

ChainDatabaseAPI

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

A class representing a database for chain data. This class is derived from [HeaderDatabaseAPI](#).

```
abstract add_receipt(block_header: BlockHeaderAPI, index_key: int, receipt: ReceiptAPI) → Hash32
```

Add the given receipt to the provided block header.

Return the updated *receipts_root* for updated block header.

```
abstract add_transaction(block_header: BlockHeaderAPI, index_key: int, transaction:
                           SignedTransactionAPI) → Hash32
```

Add the given transaction to the provided block header.

Return the updated *transactions_root* for updated block header.

```
abstract exists(key: bytes) → bool
```

Return True if the given key exists in the database.

```
abstract get(key: bytes) → bytes
```

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

```
abstract get_block_transaction_hashes(block_header: BlockHeaderAPI) → Tuple[Hash32, ...]
```

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

```
abstract get_block_transactions(block_header: BlockHeaderAPI, transaction_decoder:
                                  Type[TransactionDecoderAPI]) → Tuple[SignedTransactionAPI, ...]
```

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

```
abstract get_block_uncles(uncles_hash: Hash32) → Tuple[BlockHeaderAPI, ...]
```

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

```
abstract get_block_withdrawals(block_header: BlockHeaderAPI) → Tuple[WithdrawalAPI, ...]
```

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

```
abstract get_receipt_by_index(block_number: BlockNumber, receipt_index: int, receipt_decoder:
                               Type[ReceiptDecoderAPI]) → ReceiptAPI
```

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

```
abstract get_receipts(header: BlockHeaderAPI, receipt_decoder: Type[ReceiptDecoderAPI]) →
                       Tuple[ReceiptAPI, ...]
```

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

abstract `get_transaction_by_index`(*block_number*: *BlockNumber*, *transaction_index*: *int*,
 transaction_decoder: *Type*[*TransactionDecoderAPI*]) →
 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.

abstract `get_transaction_index`(*transaction_hash*: *Hash32*) → *Tuple*[*BlockNumber*, *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.

abstract `persist_block`(*block*: *BlockAPI*, *genesis_parent_hash*: *Hash32* | *None* = *None*) →
 Tuple[*Tuple*[*Hash32*, ...], *Tuple*[*Hash32*, ...]]

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.

Warning: This API assumes all block transactions have been persisted already. Use *eth.abc.ChainDatabaseAPI.persist_unexecuted_block()* to persist blocks that were not executed.

abstract `persist_trie_data_dict`(*trie_data_dict*: *Dict*[*Hash32*, *bytes*]) → *None*

Store raw trie data to db from a dict

abstract `persist_uncles`(*uncles*: *Tuple*[*BlockHeaderAPI*]) → *Hash32*

Persist the list of uncles to the database.

Return the uncles hash.

abstract `persist_unexecuted_block`(*block*: *BlockAPI*, *receipts*: *Tuple*[*ReceiptAPI*, ...],
 genesis_parent_hash: *Hash32* | *None* = *None*) →
 Tuple[*Tuple*[*Hash32*, ...], *Tuple*[*Hash32*, ...]]

Persist the given block's header, uncles, transactions, and receipts. Does **not** validate if state transitions are valid.

Parameters

- **block** – the block that gets persisted
- **receipts** – the receipts for the given block
- **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.

This API should be used to persist blocks that the EVM does not execute but which it stores to make them available. It ensures to persist receipts and transactions which *eth.abc.ChainDatabaseAPI.persist_block()* in contrast assumes to be persisted separately.

db: *AtomicDatabaseAPI*

GasMeterAPI

class eth.abc.GasMeterAPI

A class to define a gas meter.

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

Consume amount of gas for a defined reason.

abstract **refund_gas**(*amount: int*) → None

Refund amount of gas.

abstract **return_gas**(*amount: int*) → None

Return amount of gas.

gas_refunded: int

gas_remaining: int

start_gas: int

MessageAPI

class eth.abc.MessageAPI

A message for VM computation.

code: bytes

abstract **property** **code_address:** Address

create_address: Address

data: bytes | memoryview

abstract **property** **data_as_bytes:** bytes

depth: int

gas: int

abstract **property** **is_create:** bool

is_static: bool

sender: Address

should_transfer_value: bool

abstract **property** **storage_address:** Address

to: Address

value: int

OpcodeAPI

class eth.abc.**OpcodeAPI**

A class representing an opcode.

abstract classmethod **as_opcode**(*logic_fn: Callable[[ComputationAPI], None]*, *mnemonic: str*,
gas_cost: int) → *OpcodeAPI*

Class factory method for turning vanilla functions into Opcodes.

mnemonic: *str*

TransactionContextAPI

class eth.abc.**TransactionContextAPI**(*gas_price: int*, *origin: Address*)

Immutable transaction context information that remains constant over the VM execution.

abstract **get_next_log_counter**() → *int*

Increment and return the log counter.

abstract property **blob_versioned_hashes:** *Sequence[Hash32]*

Return the blob versioned hashes of the transaction context.

abstract property **gas_price:** *int*

Return the gas price of the transaction context.

abstract property **origin:** *Address*

Return the origin of the transaction context.

MemoryAPI

class eth.abc.**MemoryAPI**

A class representing the memory of the *VirtualMachineAPI*.

abstract **copy**(*destination: int*, *source: int*, *length: int*) → *bytes*

Copy bytes of memory with size *length* from *source* to *destination*

abstract **extend**(*start_position: int*, *size: int*) → *None*

Extend the memory from the given *start_position* to the provided *size*.

abstract **read**(*start_position: int*, *size: int*) → *memoryview*

Return a view into the memory

abstract **read_bytes**(*start_position: int*, *size: int*) → *bytes*

Read a value from memory and return a fresh bytes instance

abstract **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*.

abstract `dup(position: int) → None`

Perform a DUP operation on the stack.

abstract `pop1_any() → 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.

abstract `pop1_bytes() → bytes`

Pop and return a bytes element from the stack.

Raise *eth.exceptions.InsufficientStack* if the stack was empty.

abstract `pop1_int() → int`

Pop and return an integer from the stack.

Raise *eth.exceptions.InsufficientStack* if the stack was empty.

abstract `pop_any(num_items: int) → Tuple[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.

abstract `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.

abstract `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.

abstract `push_bytes(value: bytes) → None`

Push a bytes item onto the stack.

abstract `push_int(value: int) → None`

Push an integer item onto the stack.

abstract `swap(position: int) → None`

Perform a SWAP operation on the stack.

CodeStreamAPI

class eth.abc.CodeStreamAPI

A class representing a stream of EVM code.

abstract `is_valid_opcode(position: int) → bool`

Return True if a valid opcode exists at position.

abstract `peek() → int`

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

abstract `read(size: int) → bytes`

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

abstract `seek(program_counter: int) → AbstractContextManager[CodeStreamAPI]`

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

program_counter: `int`

StackManipulationAPI

class eth.abc.StackManipulationAPI

abstract `stack_pop1_any() → int | bytes`

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

abstract `stack_pop1_bytes() → bytes`

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

abstract `stack_pop1_int() → int`

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

abstract `stack_pop_any(num_items: int) → Tuple[int | bytes, ...]`

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

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

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

abstract `stack_pop_ints(num_items: int) → Tuple[int, ...]`

Pop the last num_items from the stack, returning a tuple of their ordinal values.

abstract `stack_push_bytes(value: bytes) → None`

Push value on the stack which must be a 32 byte string.

abstract `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.

abstract property base_fee_per_gas: `int | None`

Return the base fee per gas of the block

abstract property block_number: `BlockNumber`

Return the number of the block.

abstract property chain_id: `int`

Return the id of the chain.

abstract property coinbase: `Address`

Return the coinbase address of the block.

abstract property difficulty: `int`

Return the difficulty of the block.

abstract property excess_blob_gas: `int | None`

Return the excess blob gas of the block

abstract property gas_limit: `int`

Return the gas limit of the block.

abstract property mix_hash: `Hash32`

Return the mix hash of the block

abstract property prev_hashes: `Iterable[Hash32]`

Return an iterable of block hashes that precede the block.

abstract property timestamp: `int`

Return the timestamp of the block.

ComputationAPI

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

The base abstract class for all execution computations.

abstract add_child_computation(*child_computation: ComputationAPI*) \rightarrow `None`

Add the given child_computation.

abstract add_log_entry(*account: Address, topics: Tuple[int, ...], data: bytes*) \rightarrow `None`

Add a log entry.

abstract apply_child_computation(*child_msg: MessageAPI*) \rightarrow *ComputationAPI*

Apply the vm message child_msg as a child computation.

abstract classmethod apply_computation(*state: StateAPI, message: MessageAPI, transaction_context: TransactionContextAPI*) \rightarrow *ComputationAPI*

Execute the logic within the message: Either run the precompile, or step through each opcode. Generally, the only VM-specific logic is for each opcode as it executes.

This should rarely be called directly, because it will skip over other important VM-specific logic that happens before or after the execution.

Instead, prefer `apply_message()` or `apply_create_message()`.

```
abstract classmethod apply_create_message(state: StateAPI, message: MessageAPI,  
                                           transaction_context: TransactionContextAPI,  
                                           parent_computation: ComputationAPI | None = None)  
                                           → ComputationAPI
```

Execute a VM message to create a new contract. This is where the VM-specific create logic exists.

```
abstract classmethod apply_message(state: StateAPI, message: MessageAPI, transaction_context:  
                                     TransactionContextAPI, parent_computation: ComputationAPI |  
                                     None = None) → ComputationAPI
```

Execute a VM message. This is where the VM-specific call logic exists.

```
abstract 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.

```
abstract 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.

```
abstract generate_child_computation(child_msg: MessageAPI) → ComputationAPI
```

Generate a child computation from the given `child_msg`.

```
abstract get_accounts_for_deletion() → List[Address]
```

Return a tuple of addresses that are registered for deletion.

```
abstract get_gas_meter() → GasMeterAPI
```

Return the gas meter for the computation.

```
abstract get_gas_refund() → int
```

Return the number of refunded gas.

```
abstract get_gas_remaining() → int
```

Return the number of remaining gas.

```
abstract get_gas_used() → int
```

Return the number of used gas.

```
abstract 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.

```
abstract get_opcode_fn(opcode: int) → OpcodeAPI
```

Return the function for the given opcode.

```
abstract classmethod get_precompiles() → Dict[Address, Callable[[ComputationAPI], None]]
```

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

```
abstract get_raw_log_entries() → Tuple[Tuple[int, bytes, Tuple[int, ...], bytes], ...]
```

Return a tuple of raw log entries.

abstract `get_self_destruct_beneficiaries()` → `List[Address]`

Return a list of addresses that were beneficiaries of the self-destruct opcode - whether or not the contract was self-destructed, post-Cancun.

abstract `memory_copy(destination: int, source: int, length: int)` → `bytes`

Copy bytes of memory with size `length` from `source` to `destination`

abstract `memory_read_bytes(start_position: int, size: int)` → `bytes`

Read and return `size` bytes from memory starting at `start_position`.

abstract `memory_write(start_position: int, size: int, value: bytes)` → `None`

Write value to memory at `start_position`. Require that `len(value) == size`.

abstract `prepare_child_message(gas: int, to: Address, value: int, data: bytes | memoryview, code: bytes, **kwargs: Any)` → `MessageAPI`

Helper method for creating a child computation.

abstract `raise_if_error()` → `None`

If there was an error during computation, raise it as an exception immediately.

Raises

`VMError` –

abstract `refund_gas(amount: int)` → `None`

Add amount of gas to the pool of gas marked to be refunded.

abstract `register_account_for_deletion(beneficiary: Address)` → `None`

Register the address of `beneficiary` for deletion.

abstract `return_gas(amount: int)` → `None`

Return amount of gas to the available gas pool.

abstract `stack_dup(position: int)` → `None`

Duplicate the stack item at `position` and pushes it onto the stack.

abstract `stack_swap(position: int)` → `None`

Swap the item on the top of the stack with the item at `position`.

`accounts_to_delete:` `List[Address]`

`beneficiaries:` `List[Address]`

`children:` `List[ComputationAPI]`

`code:` `CodeStreamAPI`

`contracts_created:` `List[Address]` = []

abstract `property error:` `VMError`

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

abstract `property is_error:` `bool`

Return True if the computation resulted in an error.

abstract `property is_origin_computation:` `bool`

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

abstract `property is_success:` `bool`

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

logger: `ExtendedDebugLogger`

msg: `MessageAPI`

opcodes: `Dict[int, OpcodeAPI]`

abstract property output: `bytes`
Get the return value of the computation.

abstract property precompiles: `Dict[Address, Callable[[ComputationAPI], None]]`
Return a dictionary where the keys are the addresses of precompiles and the values are the precompile functions.

return_data: `bytes = b''`

abstract property should_burn_gas: `bool`
Return True if the remaining gas should be burned.

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

abstract property should_return_gas: `bool`
Return True if the remaining gas should be returned.

state: `StateAPI`

transaction_context: `TransactionContextAPI`

AccountStorageDatabaseAPI

class `eth.abc.AccountStorageDatabaseAPI`

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

abstract `commit(checkpoint: JournalDBCheckpoint) → None`

Collapse changes into the given checkpoint.

abstract `delete() → None`

Delete the entire storage at the account.

abstract `discard(checkpoint: JournalDBCheckpoint) → None`

Discard the given checkpoint.

abstract `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`.

abstract `get_accessed_slots() → FrozenSet[int]`

List all the slots that had been accessed since object creation.

abstract `get_changed_root() → Hash32`

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

abstract `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.

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

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

abstract record(*checkpoint: JournalDBCheckpoint*) → *None*
 Record changes into the given checkpoint.

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

abstract property has_changed_root: bool
 Return True if the storage root has changed.

AccountDatabaseAPI

```
class eth.abc.AccountDatabaseAPI(db: AtomicDatabaseAPI, state_root: Hash32 =
    b'\xe8x1f\x17\x1b\xccU\xa6\xff\x83E\xe6\x92\xc0\xcf8n[H\xe0x1b\x99\xad\xc0x01b\xb5
```

A class representing a database for accounts.

abstract account_exists(*address: Address*) → *bool*
 Return True if an account exists at *address*, otherwise False.

abstract account_has_code_or_nonce(*address: Address*) → *bool*
 Return True if either code or a nonce exists at *address*.

abstract account_is_empty(*address: Address*) → *bool*
 Return True if an account exists at *address*.

abstract commit(*checkpoint: JournalDBCheckpoint*) → *None*
 Collapse changes into checkpoint.

abstract delete_account(*address: Address*) → *None*
 Delete the account at *address*.

abstract delete_code(*address: Address*) → *None*
 Delete the code at *address*.

abstract delete_storage(*address: Address*) → *None*
 Delete the storage at *address*.

abstract discard(*checkpoint: JournalDBCheckpoint*) → *None*
 Discard the given checkpoint.

abstract get_balance(*address: Address*) → *int*
 Return the balance at *address*.

abstract get_code(*address: Address*) → *bytes*
 Return the code at the given *address*.

abstract get_code_hash(*address: Address*) → *Hash32*
 Return the hash of the code at *address*.

abstract get_nonce(*address: Address*) → *int*
 Return the nonce for *address*.

abstract `get_storage(address: Address, 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`.

abstract `has_root(state_root: bytes) → bool`

Return `True` if the `state_root` exists, otherwise `False`.

abstract `increment_nonce(address: Address) → None`

Increment the nonce for `address`.

abstract `is_address_warm(address: Address) → bool`

Was the account accessed during this transaction?

See EIP-2929

abstract `is_storage_warm(address: Address, slot: int) → bool`

Was the storage slot accessed during this transaction?

See EIP-2929

abstract `lock_changes() → None`

Locks in changes across all accounts' storage databases.

This is typically used at the end of a transaction, to make sure that a revert doesn't roll back through the previous transaction, and to be able to look up the "original" value of any account storage, where "original" is the beginning of a transaction (instead of the beginning of a block).

See `eth.abc.AccountStorageDatabaseAPI.lock_changes()` for what is called on each account's storage database.

abstract `make_state_root() → Hash32`

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

abstract `mark_address_warm(address: Address) → None`

Mark the account as accessed during this transaction.

See EIP-2929

abstract `mark_storage_warm(address: Address, slot: int) → None`

Mark the storage slot as accessed during this transaction.

See EIP-2929

abstract `persist() → MetaWitnessAPI`

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`.

abstract record() → `JournalDBCheckpoint`
 Create and return a new checkpoint.

abstract set_balance(*address: Address, balance: int*) → `None`
 Set balance as the new balance for *address*.

abstract set_code(*address: Address, code: bytes*) → `None`
 Set code as the new code at *address*.

abstract set_nonce(*address: Address, nonce: int*) → `None`
 Set nonce as the new nonce for *address*.

abstract set_storage(*address: Address, slot: int, value: int*) → `None`
 Write value into slot for the given *address*.

abstract touch_account(*address: Address*) → `None`
 Touch the account at *address*.

abstract property state_root: `Hash32`
 Return the state root hash.

TransactionExecutorAPI

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

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

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

abstract calc_data_fee(*transaction: BlobTransaction*) → `int`
 For Cancun and later, calculate the data fee for a transaction.

abstract finalize_computation(*transaction: SignedTransactionAPI, computation: ComputationAPI*) → `ComputationAPI`
 Finalize the transaction.

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

ConfigurableAPI

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

abstract classmethod configure(*__name__: str | None = None, **overrides: Any*) → `Type[T]`

StateAPI

class `eth.abc.StateAPI`(*db*: `AtomicDatabaseAPI`, *execution_context*: `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.
-

abstract `account_exists`(*address*: `Address`) → `bool`

Return True if an account exists at *address*.

abstract `account_is_empty`(*address*: `Address`) → `bool`

Return True if the account at *address* is empty, otherwise False.

abstract `apply_all_withdrawals`(*withdrawals*: `Sequence[WithdrawalAPI]`) → `None`

abstract `apply_transaction`(*transaction*: `SignedTransactionAPI`) → `ComputationAPI`

Apply transaction to the vm state

Parameters

transaction – the transaction to apply

Returns

the computation

abstract `apply_withdrawal`(*withdrawal*: `WithdrawalAPI`) → `None`

abstract `clear_transient_storage`() → `None`

Clear the transient storage. Should be done at the start of every transaction

abstract `commit`(*snapshot*: `Tuple[Hash32, JournalDBCcheckpoint]`) → `None`

Commit the journal to the point where the snapshot was taken. This merges in any changes that were recorded since the snapshot.

abstract `costless_execute_transaction`(*transaction*: `SignedTransactionAPI`) → `ComputationAPI`

Execute the given transaction with a gas price of 0.

abstract `delete_account`(*address*: `Address`) → `None`

Delete the account at the given address.

abstract `delete_code`(*address*: `Address`) → `None`

Delete the code at address.

abstract `delete_storage`(*address*: `Address`) → `None`

Delete the storage at address

abstract `delta_balance`(*address*: `Address`, *delta*: `int`) → `None`

Apply delta to the balance at address.

abstract classmethod `get_account_db_class`() → `Type[AccountDatabaseAPI]`

Return the *AccountDatabaseAPI* class that the state class uses.

abstract `get_ancestor_hash(block_number: BlockNumber) → Hash32`

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).

abstract `get_balance(address: Address) → int`

Return the balance for the account at `address`.

abstract `get_code(address: Address) → bytes`

Return the code at `address`.

abstract `get_code_hash(address: Address) → Hash32`

Return the hash of the code at `address`.

abstract `get_computation(message: MessageAPI, transaction_context: TransactionContextAPI) → ComputationAPI`

Return a computation instance for the given `message` and `transaction_context`

abstract `get_gas_price(transaction: SignedTransactionAPI) → int`

Return the gas price of the given transaction.

Factor in the current block's base gas price, if appropriate. (See EIP-1559)

abstract `get_nonce(address: Address) → int`

Return the nonce at `address`.

abstract `get_storage(address: Address, slot: int, from_journal: bool = True) → int`

Return the storage at `slot` for `address`.

abstract `get_tip(transaction: SignedTransactionAPI) → int`

Return the gas price that gets allocated to the miner/validator.

Pre-EIP-1559 that would be the full transaction gas price. After, it would be the tip price (potentially reduced, if the base fee is so high that it surpasses the transaction's maximum gas price after adding the tip).

abstract `get_transaction_context(transaction: SignedTransactionAPI) → TransactionContextAPI`

Return the `TransactionContextAPI` for the given transaction

abstract classmethod `get_transaction_context_class() → Type[TransactionContextAPI]`

Return the `BaseTransactionContext` class that the state class uses.

abstract `get_transaction_executor() → TransactionExecutorAPI`

Return the transaction executor.

abstract `get_transient_storage(address: Address, slot: int) → bytes`

Return the transient storage for `address` at slot `slot`.

abstract `has_code_or_nonce(address: Address) → bool`

Return True if either a nonce or code exists at the given address.

abstract `increment_nonce(address: Address) → None`

Increment the nonce at `address`.

abstract `is_address_warm(address: Address) → bool`

Was the account accessed during this transaction?

See EIP-2929

abstract is_storage_warm(*address: Address, slot: int*) → bool

Was the storage slot accessed during this transaction?

See EIP-2929

abstract 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.

abstract make_state_root() → Hash32

Create and return the state root.

abstract mark_address_warm(*address: Address*) → None

Mark the account as accessed during this transaction.

See EIP-2929

abstract mark_storage_warm(*address: Address, slot: int*) → None

Mark the storage slot as accessed during this transaction.

See EIP-2929

abstract override_transaction_context(*gas_price: int*) → AbstractContextManager[None]

Return a [ContextManager](#) that overwrites the current transaction context, applying the given *gas_price*.

abstract persist() → MetaWitnessAPI

Persist the current state to the database.

abstract revert(*snapshot: Tuple[Hash32, JournalDBCheckpoint]*) → None

Revert the VM to the state at the snapshot

abstract set_balance(*address: Address, balance: int*) → None

Set balance to the balance at *address*.

abstract set_code(*address: Address, code: bytes*) → None

Set code as the new code at *address*.

abstract set_nonce(*address: Address, nonce: int*) → None

Set nonce as the new nonce at *address*.

abstract set_storage(*address: Address, slot: int, value: int*) → None

Write value to the given slot at *address*.

abstract set_transient_storage(*address: Address, slot: int, value: bytes*) → None

Return the transient storage for *address* at slot *slot*.

abstract snapshot() → Tuple[Hash32, JournalDBCheckpoint]

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.

abstract touch_account(*address: Address*) → None

Touch the account at the given *address*.

abstract validate_transaction(*transaction: SignedTransactionAPI*) → None

Validate the given transaction.

account_db_class: `Type[AccountDatabaseAPI]`

abstract property base_fee: `int`
 Return the current `base_fee` from the current `execution_context`
 Raises a `NotImplementedError` if called in an execution context prior to the London hard fork.

abstract property blob_base_fee: `int`
 Return the current `blob_base_fee` from the current `execution_context`
 Raises a `NotImplementedError` if called in an execution context prior to the Cancun hard fork.

abstract property block_number: `BlockNumber`
 Return the current `block_number` from the current `execution_context`

abstract property coinbase: `Address`
 Return the current `coinbase` from the current `execution_context`

computation_class: `Type[ComputationAPI]`

abstract property difficulty: `int`
 Return the current `difficulty` from the current `execution_context`

execution_context: `ExecutionContextAPI`

abstract property gas_limit: `int`
 Return the current `gas_limit` from the current `transaction_context`

abstract property logger: `ExtendedDebugLogger`
 Return the logger.

abstract property mix_hash: `Hash32`
 Return the current `mix_hash` from the current `execution_context`

abstract property state_root: `Hash32`
 Return the current `state_root` from the underlying database

abstract property timestamp: `int`
 Return the current `timestamp` from the current `execution_context`

transaction_context_class: `Type[TransactionContextAPI]`

transaction_executor_class: `Type[TransactionExecutorAPI] = None`

VirtualMachineAPI

class `eth.abc.VirtualMachineAPI`(*header: BlockHeaderAPI, chaindb: ChainDatabaseAPI, chain_context: ChainContextAPI, consensus_context: 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.

abstract add_receipt_to_header(*old_header*: [BlockHeaderAPI](#), *receipt*: [ReceiptAPI](#)) → [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.

abstract apply_all_transactions(*transactions*: [Sequence](#)[[SignedTransactionAPI](#)], *base_header*: [BlockHeaderAPI](#)) → [Tuple](#)[[BlockHeaderAPI](#), [Tuple](#)[[ReceiptAPI](#), ...], [Tuple](#)[[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_all_withdrawals(*withdrawals*: [Sequence](#)[[WithdrawalAPI](#)]) → [None](#)

Updates the state by applying all withdrawals. This does *not* update the current block or header of the VM.

Parameters

- **withdrawals** – an iterable of all withdrawals to apply

abstract apply_transaction(*header*: [BlockHeaderAPI](#), *transaction*: [SignedTransactionAPI](#)) → [Tuple](#)[[ReceiptAPI](#), [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

abstract classmethod build_state(*db*: [AtomicDatabaseAPI](#), *header*: [BlockHeaderAPI](#), *chain_context*: [ChainContextAPI](#), *previous_hashes*: [Iterable](#)[[Hash32](#)] = ()) → [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.

abstract classmethod compute_difficulty(*parent_header*: [BlockHeaderAPI](#), *timestamp*: [int](#)) → [int](#)

Compute the difficulty for a block header.

Parameters

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

abstract configure_header(***header_params*: [Any](#)) → [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.

abstract static create_execution_context(*header: BlockHeaderAPI, prev_hashes: Iterable[Hash32], chain_context: ChainContextAPI*) → *ExecutionContextAPI*

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

abstract classmethod create_genesis_header(***genesis_params: Any*) → *BlockHeaderAPI*

Create a genesis header using this VM's rules.

This is equivalent to calling *create_header_from_parent()* with *parent_header* set to *None*.

abstract classmethod create_header_from_parent(*parent_header: BlockHeaderAPI, **header_params: Any*) → *BlockHeaderAPI*

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

abstract create_transaction(**args: Any, **kwargs: Any*) → *SignedTransactionAPI*

Proxy for instantiating a signed transaction for this VM.

abstract classmethod create_unsigned_transaction(**, nonce: int, gas_price: int, gas: int, to: Address, value: int, data: bytes*) → *UnsignedTransactionAPI*

Proxy for instantiating an unsigned transaction for this VM.

abstract execute_bytecode(*origin: Address, gas_price: int, gas: int, to: Address, sender: Address, value: int, data: bytes, code: bytes, code_address: Address | None = None*) → *ComputationAPI*

Execute raw bytecode in the context of the current state of the virtual machine. Note that this skips over some of the logic that would normally happen during a call. Watch out for:

- value (ether) is *not* transferred
- state is *not* rolled back in case of an error
- The target account is *not* necessarily created
- others...

For other potential surprises, check the implementation differences between *ComputationAPI.apply_computation()* and *ComputationAPI.apply_message()*. (depending on the VM fork)

abstract finalize_block(*block: BlockAPI*) → *BlockAndMetaWitness*

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

abstract classmethod generate_block_from_parent_header_and_coinbase(*parent_header: BlockHeaderAPI, coinbase: Address*) → *BlockAPI*

Generate block from parent header and coinbase.

abstract get_block() → *BlockAPI*

Return the current block.

abstract classmethod get_block_class() → *Type[BlockAPI]*

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

abstract 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

abstract `get_header()` → *BlockHeaderAPI*

Return the current header.

abstract 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

abstract classmethod `get_prev_hashes(last_block_hash: Hash32, chaindb: ChainDatabaseAPI) → Iterable[Hash32] | None`

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

abstract classmethod `get_receipt_builder()` → *Type[ReceiptBuilderAPI]*

Return the class that this VM uses to encode and decode receipts.

abstract classmethod `get_state_class()` → *Type[StateAPI]*

Return the class that this VM uses for states.

abstract classmethod `get_transaction_builder()` → *Type[TransactionBuilderAPI]*

Return the class that this VM uses to build and encode transactions.

abstract static `get_uncle_reward(block_number: BlockNumber, uncle: BlockHeaderAPI) → 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

abstract `import_block(block: BlockAPI) → BlockAndMetaWitness`

Import the given block to the chain.

abstract `in_costless_state()` → *AbstractContextManager[StateAPI]*

Return a *ContextManager* with the current state wrapped in a temporary block. In this state, the ability to pay gas costs is ignored.

abstract `increment_blob_gas_used(old_header: BlockHeaderAPI, transaction: TransactionFieldsAPI) → BlockHeaderAPI`

Update the header by incrementing the `blob_gas_used` for the transaction.

abstract `make_receipt(base_header: BlockHeaderAPI, transaction: SignedTransactionAPI, computation: ComputationAPI, state: StateAPI) → 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

abstract mine_block(*block*: BlockAPI, *args: Any, **kwargs: Any) → BlockAndMetaWitness

Mine the given block. Proxies to self.pack_block method.

abstract pack_block(*block*: BlockAPI, *args: Any, **kwargs: Any) → 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

abstract set_block_transactions_and_withdrawals(*base_block*: BlockAPI, *new_header*: BlockHeaderAPI, *transactions*: Sequence[SignedTransactionAPI], *receipts*: Sequence[ReceiptAPI], *withdrawals*: Sequence[WithdrawalAPI] | None = None) → BlockAPI

Create a new block with the given transactions and/or withdrawals.

transaction_applied_hook(*transaction_index*: int, *transactions*: Sequence[SignedTransactionAPI], *base_header*: BlockHeaderAPI, *partial_header*: BlockHeaderAPI, *computation*: ComputationAPI, *receipt*: ReceiptAPI) → None

A hook for a subclass to use as a way to note that a transaction was applied. This only gets triggered as part of *apply_all_transactions*, which is called by *block_import*.

abstract validate_block(*block*: BlockAPI) → None

Validate the given block.

abstract classmethod validate_header(*header*: BlockHeaderAPI, *parent_header*: BlockHeaderAPI) → None

Raises

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

abstract classmethod validate_receipt(*receipt*: ReceiptAPI) → None

Validate the given receipt.

abstract validate_seal(*header*: BlockHeaderAPI) → None

Validate the seal on the given header.

abstract validate_seal_extension(*header*: BlockHeaderAPI, *parents*: Iterable[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*.

abstract validate_transaction_against_header(*base_header*: [BlockHeaderAPI](#), *transaction*: [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

abstract classmethod validate_uncle(*block*: [BlockAPI](#), *uncle*: [BlockHeaderAPI](#), *uncle_parent*: [BlockHeaderAPI](#)) → None

Validate the given uncle in the context of the given block.

chaindb: [ChainDatabaseAPI](#)

consensus_class: [Type](#)[[ConsensusAPI](#)]

consensus_context: [ConsensusContextAPI](#)

extra_data_max_bytes: [ClassVar](#)[[int](#)]

fork: [str](#)

abstract property previous_hashes: [Iterable](#)[[Hash32](#)] | None

Convenience API for accessing the previous 255 block hashes.

abstract property state: [StateAPI](#)

Return the current state.

HeaderChainAPI

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

Like [eth.abc.ChainAPI](#) but does only support headers, not entire blocks.

abstract classmethod from_genesis_header(*base_db*: [AtomicDatabaseAPI](#), *genesis_header*: [BlockHeaderAPI](#)) → [HeaderChainAPI](#)

Initialize the chain from the genesis header.

abstract get_block_header_by_hash(*block_hash*: [Hash32](#)) → [BlockHeaderAPI](#)

Direct passthrough to *headerdb*

get_canonical_block_hash(*block_number*: [BlockNumber](#)) → [Hash32](#)

Direct passthrough to *headerdb*

abstract get_canonical_block_header_by_number(*block_number*: [BlockNumber](#)) → [BlockHeaderAPI](#)

Direct passthrough to *headerdb*

abstract get_canonical_head() → [BlockHeaderAPI](#)

Direct passthrough to *headerdb*

abstract classmethod get_headerdb_class() → [Type](#)[[HeaderDatabaseAPI](#)]

Return the class which should be used for the *headerdb*

abstract header_exists(*block_hash*: *Hash32*) → *bool*

Direct passthrough to *headerdb*

abstract import_header(*header*: *BlockHeaderAPI*) → *Tuple*[*Tuple*[*BlockHeaderAPI*, ...],
Tuple[*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.

chain_id: *int*

header: *BlockHeaderAPI*

vm_configuration: *Tuple*[*Tuple*[*BlockNumber*, *Type*[*VirtualMachineAPI*]], ...]

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.

abstract build_block_with_transactions_and_withdrawals(*transactions*:
Tuple[*SignedTransactionAPI*, ...],
parent_header: *BlockHeaderAPI* |
None = *None*, *withdrawals*:
Tuple[*WithdrawalAPI*, ...] | *None* =
None) → *Tuple*[*BlockAPI*,
Tuple[*ReceiptAPI*, ...],
Tuple[*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 into the block
- **parent_header** – parent of the new block – or canonical head if *None*
- **withdrawals** – an iterable of withdrawals to insert into the block

Returns

(new block, receipts, computations)

abstract create_header_from_parent(*parent_header*: *BlockHeaderAPI*, ***header_params*: *int* | *None*
| *BlockNumber* | *bytes* | *Address* | *Hash32*) → *BlockHeaderAPI*

Passthrough helper to the VM class of the block descending from the given header.

abstract `create_transaction(*args: Any, **kwargs: Any) → SignedTransactionAPI`

Passthrough helper to the current VM class.

abstract `create_unsigned_transaction(*, nonce: int, gas_price: int, gas: int, to: Address, value: int, data: bytes) → UnsignedTransactionAPI`

Passthrough helper to the current VM class.

abstract `estimate_gas(transaction: SignedTransactionAPI, at_header: BlockHeaderAPI | None = 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`.

abstract classmethod `from_genesis(base_db: AtomicDatabaseAPI, genesis_params: Dict[str, int | None | BlockNumber | bytes | Address | Hash32], genesis_state: Dict[Address, AccountDetails] | None = None) → ChainAPI`

Initialize the Chain from a genesis state.

abstract classmethod `from_genesis_header(base_db: AtomicDatabaseAPI, genesis_header: BlockHeaderAPI) → ChainAPI`

Initialize the chain from the genesis header.

abstract `get_ancestors(limit: int, header: BlockHeaderAPI) → Tuple[BlockAPI, ...]`

Return *limit* number of ancestor blocks from the current canonical head.

abstract `get_block() → BlockAPI`

Return the current block at the tip of the chain.

abstract `get_block_by_hash(block_hash: Hash32) → BlockAPI`

Return the requested block as specified by `block_hash`.

Raises

- `eth.exceptions.HeaderNotFound` – if the header is missing
- `eth.exceptions.BlockNotFound` – if any part of the block body is missing

abstract `get_block_by_header(block_header: BlockHeaderAPI) → BlockAPI`

Return the requested block as specified by the `block_header`.

Raises

- `eth.exceptions.BlockNotFound` – if any part of the block body is missing

abstract `get_block_header_by_hash(block_hash: Hash32) → 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.

abstract `get_canonical_block_by_number(block_number: BlockNumber) → 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.

abstract `get_canonical_block_hash(block_number: BlockNumber) → Hash32`

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.

abstract `get_canonical_block_header_by_number(block_number: BlockNumber) → 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.

abstract `get_canonical_head()` → *BlockHeaderAPI*

Return the block header at the canonical chain head.

Raise `CanonicalHeadNotFound` if there's no head defined for the canonical chain.

abstract `get_canonical_transaction(transaction_hash: Hash32)` → *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.

abstract `get_canonical_transaction_by_index(block_number: BlockNumber, index: int)` → *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.

abstract `get_canonical_transaction_index(transaction_hash: Hash32)` → *Tuple*[*BlockNumber*, *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.

abstract classmethod `get_chaindb_class()` → *Type*[*ChainDatabaseAPI*]

Return the class for the used *ChainDatabaseAPI*.

abstract `get_score(block_hash: Hash32)` → *int*

Return the difficulty score of the block with the given `block_hash`.

Raise `HeaderNotFound` if there is no matching block hash.

abstract `get_transaction_receipt(transaction_hash: Hash32)` → *ReceiptAPI*

Return the requested receipt for the transaction as specified by the `transaction_hash`.

Raise `ReceiptNotFound` if no receipt for the specified `transaction_hash` is found in the canonical chain.

abstract `get_transaction_receipt_by_index(block_number: BlockNumber, index: int)` → *ReceiptAPI*

Return the requested receipt for the transaction as specified by the `block_number` and `index`.

Raise `ReceiptNotFound` if no receipt for the specified `block_number` and `index` is found in the canonical chain.

abstract `get_transaction_result(transaction: SignedTransactionAPI, at_header: BlockHeaderAPI)` → *bytes*

Return the result of running the given transaction. This is referred to as a `call()` in web3.

abstract `get_vm(header: BlockHeaderAPI | None = None)` → *VirtualMachineAPI*

Return the VM instance for the given `header`.

abstract classmethod `get_vm_class(header: BlockHeaderAPI)` → *Type*[*VirtualMachineAPI*]

Return the VM class for the given `header`

classmethod `get_vm_class_for_block_number(block_number: BlockNumber)` → *Type*[*VirtualMachineAPI*]

Return the VM class for the given `block_number`

abstract import_block(*block*: [BlockAPI](#), *perform_validation*: *bool* = *True*) → [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.

abstract validate_block(*block*: [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.

abstract validate_chain(*root*: [BlockHeaderAPI](#), *descendants*: [Tuple](#)[[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`.

abstract validate_chain_extension(*headers*: [Tuple](#)[[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.

abstract validate_receipt(*receipt*: [ReceiptAPI](#), *at_header*: [BlockHeaderAPI](#)) → [None](#)

Validate the given receipt at the given header.

abstract validate_seal(*header*: [BlockHeaderAPI](#)) → [None](#)

Validate the seal on the given header.

abstract validate_uncles(*block*: [BlockAPI](#)) → [None](#)

Validate the uncles for the given block.

chain_id: [int](#)

chaindb: [ChainDatabaseAPI](#)

consensus_context_class: [Type](#)[[ConsensusContextAPI](#)]

vm_configuration: [Tuple](#)[[Tuple](#)[[BlockNumber](#), [Type](#)[[VirtualMachineAPI](#)]], ...]

MiningChainAPI

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

Like [ChainAPI](#) but with APIs to create blocks incrementally.

abstract apply_transaction(*transaction*: [SignedTransactionAPI](#)) → [Tuple](#)[[BlockAPI](#), [ReceiptAPI](#), [ComputationAPI](#)]

Apply the transaction to the current tip block.

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

abstract mine_all(*transactions: Sequence[SignedTransactionAPI], *args: Any, parent_header: BlockHeaderAPI | None = None, **kwargs: Any*) → *Tuple[BlockImportResult, Tuple[ReceiptAPI, ...], Tuple[ComputationAPI, ...]]*

Build a block with the given transactions, and mine it.

Optionally, supply the parent block header to mine on top of.

This is much faster than individually running `apply_transaction()` and then `mine_block()`.

abstract mine_block(**args: Any, **kwargs: Any*) → *BlockAPI*

Mines the current block. Proxies to the current Virtual Machine. See VM. `mine_block()`

abstract mine_block_extended(**args: Any, **kwargs: Any*) → *BlockAndMetaWitness*

Just like `mine_block()`, but includes extra returned info. Currently, the only extra info returned is the `MetaWitness`.

abstract set_header_timestamp(*timestamp: int*) → *None*

Set the timestamp of the pending header to mine.

This is mostly useful for testing, as the timestamp will be chosen automatically if this method is not called.

header: *BlockHeaderAPI*

4.5.2 Chain

BaseChain

class `eth.chains.base.BaseChain`

The base class for all Chain objects

classmethod `get_vm_class`(*header: BlockHeaderAPI*) → *Type[VirtualMachineAPI]*

Return the VM class for the given header

classmethod `get_vm_class_for_block_number`(*block_number: BlockNumber*) → *Type[VirtualMachineAPI]*

Return the VM class for the given `block_number`

validate_chain(*root: BlockHeaderAPI, descendants: Tuple[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[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_id: *int* = *None*

chaindb: *ChainDatabaseAPI* = *None*

chaindb_class: *Type[ChainDatabaseAPI]* = *None*

consensus_context_class: *Type[ConsensusContextAPI]* = *None*

```
vm_configuration: Tuple[Tuple[BlockNumber, Type[VirtualMachineAPI]], ...] = None
```

Chain

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

chaindb_class

alias of *ChainDB*

consensus_context_class

alias of *ConsensusContext*

```
build_block_with_transactions_and_withdrawals(transactions: Sequence[SignedTransactionAPI],
parent_header: BlockHeaderAPI | None = None,
withdrawals: Sequence[WithdrawalAPI] | None =
None) → Tuple[BlockAPI, Tuple[ReceiptAPI, ...],
Tuple[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 into the block
- **parent_header** – parent of the new block – or canonical head if *None*
- **withdrawals** – an iterable of withdrawals to insert into the block

Returns

(new block, receipts, computations)

```
create_header_from_parent(parent_header: BlockHeaderAPI, **header_params: int | None |
BlockNumber | bytes | Address | Hash32) → BlockHeaderAPI
```

Passthrough helper to the VM class of the block descending from the given header.

```
create_transaction(*args: Any, **kwargs: Any) → SignedTransactionAPI
```

Passthrough helper to the current VM class.

```
create_unsigned_transaction(*, nonce: int, gas_price: int, gas: int, to: Address, value: int, data: bytes)
→ UnsignedTransactionAPI
```

Passthrough helper to the current VM class.

```
ensure_header(header: BlockHeaderAPI | None = None) → BlockHeaderAPI
```

Return header if it is not *None*, otherwise return the header of the canonical head.

```
estimate_gas(transaction: SignedTransactionAPI, at_header: BlockHeaderAPI | None = 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: AtomicDatabaseAPI, genesis_params: Dict[str, int | None |
BlockNumber | bytes | Address | Hash32], genesis_state: Dict[Address,
AccountDetails] | None = None) → BaseChain
```

Initialize the Chain from a genesis state.

```
classmethod from_genesis_header(base_db: AtomicDatabaseAPI, genesis_header: BlockHeaderAPI)
→ BaseChain
```

Initialize the chain from the genesis header.

get_ancestors(*limit*: *int*, *header*: *BlockHeaderAPI*) → *Tuple*[*BlockAPI*, ...]

Return *limit* number of ancestor blocks from the current canonical head.

get_block() → *BlockAPI*

Return the current block at the tip of the chain.

get_block_by_hash(*block_hash*: *Hash32*) → *BlockAPI*

Return the requested block as specified by *block_hash*.

Raises

- *eth.exceptions.HeaderNotFound* – if the header is missing
- *eth.exceptions.BlockNotFound* – if any part of the block body is missing

get_block_by_header(*block_header*: *BlockHeaderAPI*) → *BlockAPI*

Return the requested block as specified by the *block_header*.

Raises

- *eth.exceptions.BlockNotFound* – if any part of the block body is missing

get_block_header_by_hash(*block_hash*: *Hash32*) → *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*: *BlockNumber*) → *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*: *BlockNumber*) → *Hash32*

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*: *BlockNumber*) → *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() → *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*: *Hash32*) → *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*: *BlockNumber*, *index*: *int*) → *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*: *Hash32*) → *Tuple*[*BlockNumber*, *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[ChainDatabaseAPI]`

Return the class for the used `ChainDatabaseAPI`.

get_score(*block_hash*: `Hash32`) → `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*: `Hash32`) → `ReceiptAPI`

Return the requested receipt for the transaction as specified by the `transaction_hash`.

Raise `ReceiptNotFound` if no receipt for the specified `transaction_hash` is found in the canonical chain.

get_transaction_receipt_by_index(*block_number*: `BlockNumber`, *index*: `int`) → `ReceiptAPI`

Return the requested receipt for the transaction as specified by the `block_number` and `index`.

Raise `ReceiptNotFound` if no receipt for the specified `block_number` and `index` is found in the canonical chain.

get_transaction_result(*transaction*: `SignedTransactionAPI`, *at_header*: `BlockHeaderAPI`) → `bytes`

Return the result of running the given transaction. This is referred to as a *call()* in web3.

get_vm(*at_header*: `BlockHeaderAPI` | `None` = `None`) → `VirtualMachineAPI`

Return the VM instance for the given header.

import_block(*block*: `BlockAPI`, *perform_validation*: `bool` = `True`) → `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.

persist_block(*block*: `BlockAPI`, *perform_validation*: `bool` = `True`) → `BlockPersistResult`

validate_block(*block*: `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_receipt(*receipt*: `ReceiptAPI`, *at_header*: `BlockHeaderAPI`) → `None`

Validate the given receipt at the given header.

validate_seal(*header*: `BlockHeaderAPI`) → `None`

Validate the seal on the given header.

validate_uncles(*block*: `BlockAPI`) → `None`

Validate the uncles for the given block.

gas_estimator: `StaticMethod[Callable[[StateAPI, SignedTransactionAPI], int]]` = `None`

logger = `<Logger eth.chain.chain.Chain (WARNING)>`

MiningChain

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

apply_transaction(*transaction*: `SignedTransactionAPI`) → `Tuple`[`BlockAPI`, `ReceiptAPI`, `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*: `BlockHeaderAPI` | `None` = `None`) → `VirtualMachineAPI`

Return the VM instance for the given header.

import_block(*block*: `BlockAPI`, *perform_validation*: `bool` = `True`) → `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_all(*transactions*: `Sequence`[`SignedTransactionAPI`], **args*: `Any`, *parent_header*: `BlockHeaderAPI` | `None` = `None`, *withdrawals*: `Sequence`[`WithdrawalAPI`] | `None` = `None`, ***kwargs*: `Any`) → `Tuple`[`BlockImportResult`, `Tuple`[`ReceiptAPI`, ...], `Tuple`[`ComputationAPI`, ...]]

Build a block with the given transactions, and mine it.

Optionally, supply the parent block header to mine on top of.

This is much faster than individually running `apply_transaction()` and then `mine_block()`.

mine_block(**args*: `Any`, ***kwargs*: `Any`) → `BlockAPI`

Mine whatever transactions have been incrementally applied so far.

mine_block_extended(**args*: `Any`, ***kwargs*: `Any`) → `BlockAndMetaWitness`

Just like `mine_block()`, but includes extra returned info. Currently, the only extra info returned is the `MetaWitness`.

set_header_timestamp(*timestamp*: `int`) → `None`

Set the timestamp of the pending header to mine.

This is mostly useful for testing, as the timestamp will be chosen automatically if this method is not called.

header: `BlockHeaderAPI` = `None`

4.5.3 DataBase

KeyAccessLoggerDB

KeyAccessLoggerDB

class `eth.db.accesslog.KeyAccessLoggerDB`(*wrapped_db*: `DatabaseAPI`, *log_missing_keys*: `bool` = `True`)

Wraps around a database and tracks all the keys that were read since initialization.

property `keys_read`: `FrozenSet`[`bytes`]

```
logger = <Logger eth.db.KeyAccessLoggerDB (WARNING)>
```

Account

AccountDB

```
class eth.db.account.AccountDB(db: AtomicDatabaseAPI, state_root: Hash32 =
    b'\xe8\x1f\x17\x1b\xcc\x6\xff\x83E\xe6\x92\xc0\xf8n[H\xe0\x1b\x99\xad\xc0\x01b\xb5\xe

    account_exists(address: Address) → bool
        Return True if an account exists at address, otherwise False.

    account_has_code_or_nonce(address: Address) → bool
        Return True if either code or a nonce exists at address.

    account_is_empty(address: Address) → bool
        Return True if an account exists at address.

    commit(checkpoint: JournalDBCheckpoint) → None
        Collapse changes into checkpoint.

    delete_account(address: Address) → None
        Delete the account at address.

    delete_code(address: Address) → None
        Delete the code at address.

    delete_storage(address: Address) → None
        Delete the storage at address.

    discard(checkpoint: JournalDBCheckpoint) → None
        Discard the given checkpoint.

    get_balance(address: Address) → int
        Return the balance at address.

    get_code(address: Address) → bytes
        Return the code at the given address.

    get_code_hash(address: Address) → Hash32
        Return the hash of the code at address.

    get_nonce(address: Address) → int
        Return the nonce for address.

    get_storage(address: Address, 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: Address) → None
        Increment the nonce for address.
```

is_address_warm(*address: Address*) → bool

Was the account accessed during this transaction?

See EIP-2929

is_storage_warm(*address: Address, slot: int*) → bool

Was the storage slot accessed during this transaction?

See EIP-2929

lock_changes() → None

Locks in changes across all accounts' storage databases.

This is typically used at the end of a transaction, to make sure that a revert doesn't roll back through the previous transaction, and to be able to look up the "original" value of any account storage, where "original" is the beginning of a transaction (instead of the beginning of a block).

See [eth.abc.AccountStorageDatabaseAPI.lock_changes\(\)](#) for what is called on each account's storage database.

make_state_root() → Hash32

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

mark_address_warm(*address: Address*) → None

Mark the account as accessed during this transaction.

See EIP-2929

mark_storage_warm(*address: Address, slot: int*) → None

Mark the storage slot as accessed during this transaction.

See EIP-2929

persist() → MetaWitnessAPI

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() → JournalDBCCheckpoint

Create and return a new checkpoint.

set_balance(*address: Address, balance: int*) → None

Set balance as the new balance for address.

set_code(*address: Address, code: bytes*) → None

Set code as the new code at address.

set_nonce(*address: Address, nonce: int*) → None

Set nonce as the new nonce for address.

set_storage(*address: Address, slot: int, value: int*) → *None*

Write value into slot for the given address.

touch_account(*address: Address*) → *None*

Touch the account at address.

logger = <ExtendedDebugLogger eth.db.account.AccountDB (WARNING)>

property state_root: Hash32

Return the state root hash.

Atomic

AtomicDB

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

atomic_batch() → *Iterator*[AtomicWriteBatchAPI]

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

logger = <Logger eth.db.AtomicDB (WARNING)>

wrapped_db: *DatabaseAPI* = *None*

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

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

logger = <Logger eth.db.AtomicDBWriteBatch (WARNING)>

Backends

BaseDB

class eth.db.backends.base.**BaseDB**(*args, **kwargs)

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.

MemoryDB

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

```
    kv_store: Dict[bytes, bytes] = None
```

Batch

BatchDB

```
class eth.db.batch.BatchDB(wrapped_db: 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.

commit(*apply_deletes: bool = True*) → None

commit_to(*target_db: DatabaseAPI, apply_deletes: bool = True*) → None

diff() → *DBDiff*

logger = <Logger eth.db.BatchDB (WARNING)>

wrapped_db: *DatabaseAPI* = None

Cache

CacheDB

```
class eth.db.cache.CacheDB(db: DatabaseAPI, cache_size: int = 2048)
```

Set and get decoded RLP objects, where the underlying db stores encoded objects.

reset_cache() → None

Chain

ChainDB

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

add_receipt(*block_header: BlockHeaderAPI, index_key: int, receipt: ReceiptAPI*) → Hash32

Add the given receipt to the provided block header.

Return the updated *receipts_root* for updated block header.

add_transaction(*block_header: BlockHeaderAPI, index_key: int, transaction: SignedTransactionAPI*) → Hash32

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*: *BlockHeaderAPI*) → *Tuple*[*Hash32*, ...]

Returns an iterable of the transaction hashes from the block specified by the given block header.

get_block_transactions(*header*: *BlockHeaderAPI*, *transaction_decoder*:
Type[*TransactionDecoderAPI*]) → *Tuple*[*SignedTransactionAPI*, ...]

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

get_block_uncles(*uncles_hash*: *Hash32*) → *Tuple*[*BlockHeaderAPI*, ...]

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

get_block_withdrawals(*header*: *BlockHeaderAPI*) → *Tuple*[*WithdrawalAPI*, ...]

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

get_chain_gaps() → *Tuple*[*Tuple*[*Tuple*[*BlockNumber*, *BlockNumber*], ...], *BlockNumber*]

get_receipt_by_index(*block_number*: *BlockNumber*, *receipt_index*: *int*, *receipt_decoder*:
Type[*ReceiptDecoderAPI*]) → *ReceiptAPI*

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

get_receipts(*header*: *BlockHeaderAPI*, *receipt_decoder*: *Type*[*ReceiptDecoderAPI*]) →
Iterable[*ReceiptAPI*]

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

get_transaction_by_index(*block_number*: *BlockNumber*, *transaction_index*: *int*, *transaction_decoder*:
Type[*TransactionDecoderAPI*]) → *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*: *Hash32*) → *Tuple*[*BlockNumber*, *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*: *BlockAPI*, *genesis_parent_hash*: *Hash32* =
b'\x00'
→ *Tuple*[*Tuple*[*Hash32*, ...], *Tuple*[*Hash32*, ...]])

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.

| |
|--|
| <p>Warning: This API assumes all block transactions have been persisted already. Use <code>eth.abc.ChainDatabaseAPI.persist_unexecuted_block()</code> to persist blocks that were not executed.</p> |
|--|

pending_items() → *Iterable*[*Tuple*[bytes, bytes]]

A tuple of (key, value) pairs for every key that has been updated. Like *pending_keys()*, this does not return any deleted keys.

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.

diff() → *DBDiff*

DiffMissingError

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

property is_deleted: bool

Header

HeaderDB

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

get_block_header_by_hash(*block_hash: Hash32*) → *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: BlockNumber*) → *Hash32*

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: BlockNumber*) → *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() → *BlockHeaderAPI*

Return the current block header at the head of the chain.

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() → *DBDiff*

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

discard(*checkpoint: JournalDBCheckpoint*) → *None*

Throws away all journaled data starting at the given checkpoint

flatten() → *None*

Commit everything possible without persisting

has_checkpoint(*checkpoint: JournalDBCheckpoint*) → *bool*

has_clear() → *bool*

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.

commit

record

Schema

SchemaV1

```
class eth.db.schema.SchemaV1
```

```
    static make_block_hash_to_score_lookup_key(block_hash: Hash32) → bytes
```

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

```
    static make_block_number_to_hash_lookup_key(block_number: BlockNumber) → 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_chain_gaps_lookup_key() → bytes
```

```
    static make_checkpoint_headers_key() → bytes
```

Checkpoint header hashes stored as concatenated 32 byte values

static `make_header_chain_gaps_lookup_key()` → `bytes`

Return the lookup key to retrieve the header chain integrity info from the database.

static `make_transaction_hash_to_block_lookup_key(transaction_hash: Hash32)` → `bytes`

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

static `make_withdrawal_hash_to_block_lookup_key(withdrawal_hash: Hash32)` → `bytes`

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

Storage

AccountStorageDB

class `eth.db.storage.AccountStorageDB(db: AtomicDatabaseAPI, storage_root: Hash32, address: Address)`

commit(*checkpoint: JournalDBCheckpoint*) → `None`

Collapse changes into the given checkpoint.

delete() → `None`

Delete the entire storage at the account.

discard(*checkpoint: JournalDBCheckpoint*) → `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_accessed_slots() → `FrozenSet[int]`

List all the slots that had been accessed since object creation.

get_changed_root() → `Hash32`

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: DatabaseAPI*) → `None`

Persist all changes to the database.

record(*checkpoint: JournalDBCheckpoint*) → `None`

Record changes into the given checkpoint.

set(*slot: int, value: int*) → `None`

Write value into `slot`.

property `has_changed_root: bool`

Return `True` if the storage root has changed.

`logger = <ExtendedDebugLogger eth.db.storage.AccountStorageDB (WARNING)>`

StorageLookup

class `eth.db.storage.StorageLookup`(*db*: `DatabaseAPI`, *storage_root*: `Hash32`, *address*: `Address`)

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*: `DatabaseAPI`) → `None`

Trying to commit changes when nothing has been written will raise a `ValidationError`

get_changed_root() → `Hash32`

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

property `has_changed_root`: `bool`

logger = `<ExtendedDebugLogger eth.db.storage.StorageLookup (WARNING)>`

4.5.4 Exceptions

exception `eth.exceptions.BlockNotFound`

Raised when the block with the given number/hash does not exist. This will happen, for example, if the transactions or uncles are not saved in the database.

exception `eth.exceptions.CanonicalHeadNotFound`

Raised when the chain has no canonical head.

exception `eth.exceptions.CheckpointsMustBeCanonical`

Raised when a persisted header attempts to de-canonicalize a checkpoint

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.GapTrackingCorrupted`

Raised when the tracking of chain gaps appears to be corrupted (e.g. overlapping gaps)

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 RETURNDATACOPY)

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-evm errors.

exception eth.exceptions.ReceiptNotFound

Raised when the Receipt with the given receipt index does not exist.

exception eth.exceptions.ReservedBytesInCode

Raised when bytes for the code to be deployed are reserved for a particular reason.

exception eth.exceptions.Revert

Raised when the REVERT opcode occurred

burns_gas = False

erases_return_data = False

exception eth.exceptions.StackDepthLimit

Raised when the call stack has exceeded it's maximum allowed depth.

exception eth.exceptions.StateRootNotFound

Raised when the requested state root is not present in our DB.

property missing_state_root: Hash32

exception eth.exceptions.TransactionNotFound

Raised when the transaction with the given hash or block index does not exist.

exception eth.exceptions.UnrecognizedTransactionType

Raised when an encoded transaction is using a first byte that is valid, but unrecognized. According to EIP 2718, the byte may be in the range [0, 0x7f]. As of the Berlin hard fork, all of those versions are undefined, except for 0x01 in EIP 2930.

property type_int: int

```
exception eth.exceptions.VMError
```

Base class for errors raised during VM execution.

```
burns_gas = True
```

```
erases_return_data = True
```

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.

4.5.5 RLP

Accounts

Account

```
class eth.rlp.accounts.Account(nonce: int = 0, balance: int = 0, storage_root: bytes =
    b'\x00' * 32, code_hash: bytes =
    b'\x00' * 32, **kwargs: Any)
```

RLP object for accounts.

property balance

property code_hash

property nonce

```
property storage_root
```

Blocks

BaseBlock

```
class eth.rlp.blocks.BaseBlock(*args, **kwargs)
```

```
classmethod get_transaction_builder() → Type[TransactionBuilderInterface]
```

Return the transaction builder for the block.

```
property is_genesis: bool
```

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

```
transaction_builder: Type[TransactionBuilderAPI] = None
```

Headers

BlockHeader

```
class eth.rlp.headers.BlockHeader(**kwargs: int | None | BlockNumber | bytes | Address | Hash32)
class eth.rlp.headers.BlockHeader(difficulty: int, block_number: BlockNumber, gas_limit: int, timestamp:
    int = None, coinbase: Address = ZERO_ADDRESS, parent_hash:
    Hash32 = ZERO_HASH32, uncles_hash: Hash32 =
    EMPTY_UNCLE_HASH, state_root: Hash32 = BLANK_ROOT_HASH,
    transaction_root: Hash32 = BLANK_ROOT_HASH, receipt_root:
    Hash32 = BLANK_ROOT_HASH, bloom: int = 0, gas_used: int = 0,
    extra_data: bytes = b'', mix_hash: Hash32 = ZERO_HASH32, nonce:
    bytes = GENESIS_NONCE)

property base_fee_per_gas: int
    Return the base fee per gas of the block.
    Set to None in pre-EIP-1559 (London) header.

property blob_gas_used: int
    Return blob gas used.

property block_number

property bloom

property coinbase

property difficulty

property excess_blob_gas: int
    Return excess blob gas.

property extra_data

property gas_limit

property gas_used

property hash: Hash32
    Return the hash of the block header.

property hex_hash: str
    Return the hash as a hex string.

property is_genesis: bool
    Return True if this header represents the genesis block of the chain, otherwise False.

property mining_hash: Hash32
    Return the mining hash of the block header.

property mix_hash

property nonce

property parent_beacon_block_root: Hash32 | None
    Return the hash of the parent beacon block.
```

property `parent_hash`

property `receipt_root`

property `state_root`

property `timestamp`

property `transaction_root`

property `uncles_hash`

property `withdrawals_root`: `Hash32` | `None`

Return the withdrawals root of the block.

Set to None in pre-Shanghai header.

Logs

Log

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

    property address

    property bloomables: Tuple[bytes, ...]

    property data

    property topics
```

Receipts

Receipt

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

classmethod `decode(encoded: bytes) → ReceiptAPI`

This decodes a receipt that is encoded to either a typed receipt, a legacy receipt, or the body of a typed receipt. It assumes that typed receipts are *not* rlp-encoded first.

If dealing with an object that is always rlp encoded, then use this instead:

```
    rlp.decode(encoded, sedes=ReceiptBuilderAPI)
```

For example, you may receive a list of receipts via a devp2p request. Each receipt is either a (legacy) rlp list, or a (new-style) bytestring. Even if the receipt is a bytestring, it's wrapped in an rlp bytestring, in that context. New-style receipts will *not* be wrapped in an RLP bytestring in other contexts. They will just be an EIP-2718 type-byte plus payload of concatenated bytes, which cannot be decoded as RLP. This happens for example, when calculating the receipt root hash.

encode() → `bytes`

This encodes a receipt, no matter if it's: a legacy receipt, a typed receipt, or the payload of a typed receipt. See more context in `decode`.

```
property bloom
property bloom_filter: BloomFilter
property gas_used
property logs
property state_root
type_id = None
```

Transactions

BaseTransactionMethods

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

```
    gas_used_by(computation: 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.

    property access_list: Sequence[Tuple[Address, Sequence[int]]]
        Get addresses to be accessed by a transaction, and their storage slots.

    property chain_id: int | None

    property intrinsic_gas: int
        Convenience property for the return value of get_intrinsic_gas
```

BaseTransactionFields

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

```
    property data
    property gas
    property gas_price
        Will raise AttributeError if get or set on a 1559 transaction.
    property hash: Hash32
        Return the hash of the transaction.
    property nonce
    property r
    property s
    property to
    property v
    property value
```

BaseTransaction

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

classmethod `decode(encoded: bytes) → SignedTransactionAPI`

This decodes a transaction that is encoded to either a typed transaction or a legacy transaction, or even the payload of one of the transaction types. It assumes that typed transactions are *not* rlp-encoded first.

If dealing with an object that is rlp encoded first, then use this instead:

```
    rlp.decode(encoded, sedes=TransactionBuilderAPI)
```

For example, you may receive a list of transactions via a devp2p request. Each transaction is either a (legacy) rlp list, or a (new-style) bytestring. Even if the transaction is a bytestring, it's wrapped in an rlp bytestring, in that context. New-style transactions will *not* be wrapped in an RLP bytestring in other contexts. They will just be an EIP-2718 type-byte plus payload of concatenated bytes, which cannot be decoded as RLP. An example context for this is calculating the transaction root hash.

encode() → bytes

This encodes a transaction, no matter if it's: a legacy transaction, a typed transaction, or the payload of a typed transaction. See more context in `decode`.

property `data`

property `gas`

property `gas_price`

Will raise `AttributeError` if get or set on a 1559 transaction.

property `nonce`

property `r`

property `s`

property `to`

property `v`

In old transactions, this `v` field combines the `y_parity` bit and the chain ID. All new usages should prefer accessing those fields directly. But if you must access the original `v`, then you can cast to this API first (after checking that `type_id` is `None`).

property `value`

BaseUnsignedTransaction

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

property `data`

property `gas`

property `gas_price`

property `nonce`

property `to`

property `value`

4.5.6 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(vm_class: Type[VirtualMachineAPI] = '__no_default__', at_block: int |  
                                BlockNumber = '__no_default__', chain_class: Type[ChainAPI] =  
                                '__no_default__') → Type[ChainAPI]
```

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()`
- `muir_glacier_at()`
- `berlin_at()`

- `london_at()`
- `arrow_glacier_at()`
- `gray_glacier_at()`
- `paris_at()`
- `shanghai_at()`
- `cancun_at()`
- `latest_mainnet_at()` - whatever latest mainnet VM is

```
eth.tools.builder.chain.dao_fork_at(dao_fork_block_number: BlockNumber = '__no__default__',  
                                     chain_class: Type[ChainAPI] = '__no__default__') →  
                                     Type[ChainAPI]
```

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(chain_class: Type[ChainAPI] = '__no__default__') →  
                                         Type[ChainAPI]
```

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(chain_class: Type[ChainAPI] = '__no__default__') →  
                                         Type[ChainAPI]
```

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(chain_class: Type[ChainAPI] = '__no__default__') →  
                                         Type[ChainAPI]
```

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(class_name: str = '__no__default__', chain_class: Type[ChainAPI] =  
                              '__no__default__') → Type[ChainAPI]
```

Assign the given name to the chain class.

```
eth.tools.builder.chain.chain_id(chain_id: int = '__no__default__', chain_class: Type[ChainAPI] =  
                                  '__no__default__') → Type[ChainAPI]
```

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(chain_class: ChainAPI = '__no__default__', db: AtomicDatabaseAPI |  
                                None = None, params: Dict[str, int | None | BlockNumber | bytes | Address  
                                | Hash32] | None = None, state: Dict[Address, AccountDetails] |  
                                List[Tuple[Address, Dict[str, int | bytes | Dict[int, int]]]] | None = None)  
    → ChainAPI
```

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(chain: MiningChainAPI = '__no__default__') → MiningChainAPI`

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(block: BlockAPI = '__no__default__', chain: ChainAPI = '__no__default__') → ChainAPI`

Import the provided block into the chain.

`eth.tools.builder.chain.import_blocks(*blocks: BlockAPI) → Callable[[ChainAPI], ChainAPI]`

Variadic argument version of `import_block()`

`eth.tools.builder.chain.mine_block(chain: MiningChainAPI = '__no__default__', **kwargs: Any) → MiningChainAPI`

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(num_blocks: int = '__no__default__', chain: MiningChainAPI = '__no__default__') → MiningChainAPI`

Variadic argument version of `mine_block()`

`eth.tools.builder.chain.chain_split(*splits: Iterable[Callable[[...], Any]]) → Callable[[ChainAPI], Iterable[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(block_number: int | BlockNumber = '__no__default__', chain: MiningChainAPI = '__no__default__') → MiningChainAPI`

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],
        ]
    )
)
```

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 = 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: Dict[Address, AccountDetails] | List[Tuple[Address, Dict[str, int | bytes | Dict[int, int]]], 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(execution: Dict[str, Any] = '__no__default__', filler: Dict[str, Any] = '__no__default__')` → *Dict*[*str*, *Any*]

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 = None, networks: Any | None = None, transaction: TransactionDict | None = 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").

4.5.7 Virtual Machine

Computation

BaseComputation

```
class eth.vm.computation.BaseComputation(state: StateAPI, message: MessageAPI, transaction_context: 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 precompiled contracts.

add_child_computation(*child_computation*: *ComputationAPI*) → *None*

Add the given *child_computation*.

add_log_entry(*account*: *Address*, *topics*: *Tuple[int, ...]*, *data*: *bytes*) → *None*

Add a log entry.

apply_child_computation(*child_msg*: *MessageAPI*) → *ComputationAPI*

Apply the vm message *child_msg* as a child computation.

classmethod apply_computation(*state*: *StateAPI*, *message*: *MessageAPI*, *transaction_context*: *TransactionContextAPI*, *parent_computation*: *ComputationAPI* | *None* = *None*) → *ComputationAPI*

Execute the logic within the message: Either run the precompile, or step through each opcode. Generally, the only VM-specific logic is for each opcode as it executes.

This should rarely be called directly, because it will skip over other important VM-specific logic that happens before or after the execution.

Instead, prefer *apply_message()* or *apply_create_message()*.

classmethod apply_create_message(*state*: *StateAPI*, *message*: *MessageAPI*, *transaction_context*: *TransactionContextAPI*, *parent_computation*: *ComputationAPI* | *None* = *None*) → *ComputationAPI*

Execute a VM message to create a new contract. This is where the VM-specific create logic exists.

classmethod apply_message(*state*: *StateAPI*, *message*: *MessageAPI*, *transaction_context*: *TransactionContextAPI*, *parent_computation*: *ComputationAPI* | *None* = *None*) → *ComputationAPI*

Execute a VM message. This is where the VM-specific call logic exists.

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.

classmethod consume_initcode_gas_cost(*computation*: *ComputationAPI*) → *None*

Before starting the computation, consume initcode gas cost.

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*: *MessageAPI*) → *ComputationAPI*

Generate a child computation from the given *child_msg*.

get_accounts_for_deletion() → *List[Address]*

Return a tuple of addresses that are registered for deletion.

get_gas_meter() → *GasMeterAPI*

Return the gas meter for 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) → `OpcodeAPI`

Return the function for the given opcode.

classmethod get_precompiles() → `Dict[Address, Callable[[ComputationAPI], Any]]`

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

get_raw_log_entries() → `Tuple[Tuple[int, bytes, Tuple[int, ...], bytes], ...]`

Return a tuple of raw log entries.

get_self_destruct_beneficiaries() → `List[Address]`

Return a list of addresses that were beneficiaries of the self-destruct opcode - whether or not the contract was self-destructed, post-Cancun.

memory_copy(destination: int, source: int, length: int) → `None`

Copy bytes of memory with size `length` from `source` to `destination`

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: Address, value: int, data: bytes | memoryview, code: bytes, **kwargs: Any) → `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: Address) → `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`.

accounts_to_delete: `List[Address] = None`

beneficiaries: `List[Address] = None`

```

children: List[ComputationAPI] = None

code: CodeStreamAPI = None

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

property is_error: bool
    Return True if the computation resulted in an error.

property is_origin_computation: bool
    Return True if this computation is the outermost computation at depth == 0.

property is_success: bool
    Return True if the computation did not result in an error.

logger: ExtendedDebugLogger = <ExtendedDebugLogger
eth.vm.computation.BaseComputation (WARNING)>

msg: MessageAPI = None

opcodes: Dict[int, OpcodeAPI] = None

property output: bytes
    Get the return value of the computation.

property precompiles: Dict[Address, Callable[[ComputationAPI], Any]]
    Return a dictionary where the keys are the addresses of precompiles and the values are the precompile
    functions.

return_data: bytes = b''

property should_burn_gas: bool
    Return True if the remaining gas should be burned.

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

property should_return_gas: bool
    Return True if the remaining gas should be returned.

stack_pop1_any
stack_pop1_bytes
stack_pop1_int
stack_pop_any
stack_pop_bytes
stack_pop_ints
stack_push_bytes
stack_push_int

state: StateAPI = None

transaction_context: TransactionContextAPI = None

```

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.

    invalid_positions: Set[int]

    logger = <Logger eth.vm.CodeStream (WARNING)>

    program_counter: int

    valid_positions: Set[int]
```

ExecutionContext

```
class eth.vm.execution_context.ExecutionContext(coinbase: Address, timestamp: int, block_number:
    BlockNumber, difficulty: int, mix_hash: Hash32,
    gas_limit: int, prev_hashes: Iterable[Hash32],
    chain_id: int, base_fee_per_gas: int | None = None,
    excess_blob_gas: int | None = None)

    property base_fee_per_gas: int
        Return the base fee per gas of the block

    property block_number: BlockNumber
        Return the number of the block.

    property chain_id: int
        Return the id of the chain.

    property coinbase: Address
        Return the coinbase address of the block.

    property difficulty: int
        Return the difficulty of the block.

    property excess_blob_gas: int
        Return the excess blob gas of the block

    property gas_limit: int
        Return the gas limit of the block.

    property mix_hash: Hash32
        Return the mix hash of the block
```


property prev_hashes: `Iterable[Hash32]`

Return an iterable of block hashes that precede the block.

property timestamp: `int`

Return the timestamp of the block.

GasMeter

```
class eth.vm.gas_meter.GasMeter(start_gas: int, refund_strategy: ~typing.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.

gas_refunded: `int` = `None`

gas_remaining: `int` = `None`

logger = `<ExtendedDebugLogger eth.gas.GasMeter (WARNING)>`

start_gas: `int` = `None`

Memory

```
class eth.vm.memory.Memory
```

copy(*destination: int, source: int, length: int*) → `None`

Copy bytes of memory with size `length` from source to destination

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.

logger = `<Logger eth.vm.memory.Memory (WARNING)>`

Message

```
class eth.vm.message.Message(gas: int, to: Address, sender: Address, value: int, data: bytes | memoryview,
                             code: bytes, depth: int = 0, create_address: Address | None = None,
                             code_address: Address | None = None, should_transfer_value: bool = True,
                             is_static: bool = False)

    code: bytes

    property code_address: Address

    create_address: Address

    data: bytes | memoryview

    property data_as_bytes: bytes

    depth: int

    gas: int

    property is_create: bool

    is_static: bool

    logger = <Logger eth.vm.message.Message (WARNING)>

    sender: Address

    should_transfer_value: bool

    property storage_address: Address

    to: Address

    value: int
```

Opcode

```
class eth.vm.opcode Opcode

    classmethod as_opcode(logic_fn: Callable[[...], Any], mnemonic: str, gas_cost: int) → OpcodeAPI
        Class factory method for turning vanilla functions into Opcodes.

    gas_cost: int = None

    property logger: ExtendedDebugLogger

    mnemonic: str = None
```

VM

VM

```
class eth.vm.base.VM(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI, chain_context:
    ChainContextAPI, consensus_context: ConsensusContextAPI)
```

consensus_class

alias of PowConsensus

```
apply_all_transactions(transactions: Sequence[SignedTransactionAPI], base_header:
    BlockHeaderAPI) → Tuple[BlockHeaderAPI, Tuple[ReceiptAPI, ...],
    Tuple[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_all_withdrawals(withdrawals: Sequence[WithdrawalAPI]) → None
```

Updates the state by applying all withdrawals. This does *not* update the current block or header of the VM.

Parameters

withdrawals – an iterable of all withdrawals to apply

```
apply_transaction(header: BlockHeaderAPI, transaction: SignedTransactionAPI) → Tuple[ReceiptAPI,
    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

```
apply_withdrawal(withdrawal: WithdrawalAPI) → None
```

```
classmethod block_preprocessing(state: StateAPI, header: BlockHeaderAPI) → None
```

Process any state changes before processing a block. Pre-processing does not become relevant until the Cancun network upgrade.

```
classmethod build_state(db: AtomicDatabaseAPI, header: BlockHeaderAPI, chain_context:
    ChainContextAPI, previous_hashes: Iterable[Hash32] = ()) → 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: BlockHeaderAPI, prev_hashes: Iterable[Hash32],
    chain_context: ChainContextAPI) → ExecutionContextAPI
```

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

classmethod `create_genesis_header(**genesis_params: Any) → BlockHeaderAPI`

Create a genesis header using this VM's rules.

This is equivalent to calling `create_header_from_parent()` with `parent_header` set to `None`.

create_transaction(*args: Any, **kwargs: Any) → SignedTransactionAPI

Proxy for instantiating a signed transaction for this VM.

classmethod `create_unsigned_transaction(*, nonce: int, gas_price: int, gas: int, to: Address, value: int, data: bytes) → UnsignedTransactionAPI`

Proxy for instantiating an unsigned transaction for this VM.

execute_bytecode(origin: Address, gas_price: int, gas: int, to: Address, sender: Address, value: int, data: bytes, code: bytes, code_address: Address | None = None) → ComputationAPI

Execute raw bytecode in the context of the current state of the virtual machine. Note that this skips over some of the logic that would normally happen during a call. Watch out for:

- value (ether) is *not* transferred
- state is *not* rolled back in case of an error
- The target account is *not* necessarily created
- others...

For other potential surprises, check the implementation differences between `ComputationAPI`, `apply_computation()` and `ComputationAPI.apply_message()`. (depending on the VM fork)

finalize_block(block: BlockAPI) → BlockAndMetaWitness

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: BlockHeaderAPI, coinbase: Address) → BlockAPI`

Generate block from parent header and coinbase.

get_block() → BlockAPI

Return the current block.

classmethod `get_block_class() → Type[BlockAPI]`

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

get_header() → BlockHeaderAPI

Return the current header.

classmethod `get_receipt_builder() → Type[ReceiptBuilderAPI]`

Return the class that this VM uses to encode and decode receipts.

classmethod `get_state_class() → Type[StateAPI]`

Return the class that this VM uses for states.

classmethod `get_transaction_builder() → Type[TransactionBuilderAPI]`

Return the class that this VM uses to build and encode transactions.

import_block(block: BlockAPI) → BlockAndMetaWitness

Import the given block to the chain.

in_costless_state() → *Iterator*[*StateAPI*]

Return a *ContextManager* with the current state wrapped in a temporary block. In this state, the ability to pay gas costs is ignored.

mine_block(*block*: *BlockAPI*, **args*: *Any*, ***kwargs*: *Any*) → *BlockAndMetaWitness*

Mine the given block. Proxies to *self.pack_block* method.

pack_block(*block*: *BlockAPI*, **args*: *Any*, ***kwargs*: *Any*) → *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_and_withdrawals(*base_block*: *BlockAPI*, *new_header*: *BlockHeaderAPI*,
transactions: *Sequence*[*SignedTransactionAPI*], *receipts*:
Sequence[*ReceiptAPI*], *withdrawals*:
Sequence[*WithdrawalAPI*] | *None* = *None*) → *BlockAPI*

Create a new block with the given transactions and/or withdrawals.

validate_block(*block*: *BlockAPI*) → *None*

Validate the given block.

classmethod validate_gas(*header*: *BlockHeaderAPI*, *parent_header*: *BlockHeaderAPI*) → *None*

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

Raises

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

classmethod validate_receipt(*receipt*: *ReceiptAPI*) → *None*

Validate the given receipt.

validate_seal(*header*: *BlockHeaderAPI*) → *None*

Validate the seal on the given header.

validate_seal_extension(*header*: *BlockHeaderAPI*, *parents*: *Iterable*[*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*: *BlockAPI*, *uncle*: *BlockHeaderAPI*, *uncle_parent*:
BlockHeaderAPI) → *None*

Validate the given uncle in the context of the given block.

```
block_class: Type[BlockAPI] = None
chaindb: ChainDatabaseAPI = None
cls_logger = <Logger eth.vm.base.VM (WARNING)>
extra_data_max_bytes: ClassVar[int] = 32
fork: str = None
property logger: Logger
property previous_hashes: Iterable[Hash32] | None
    Convenience API for accessing the previous 255 block hashes.
property state: StateAPI
    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() → 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[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.

logger = <Logger eth.vm.stack.Stack (WARNING)>

values

State

BaseState

class *eth.vm.state.BaseState*(*db: AtomicDatabaseAPI, execution_context: ExecutionContextAPI, state_root: Hash32*)

account_exists(*address: Address*) → *bool*

Return True if an account exists at *address*.

account_is_empty(*address: Address*) → *bool*

Return True if the account at *address* is empty, otherwise False.

apply_all_withdrawals(*withdrawals: Sequence[WithdrawalAPI]*) → *None*

apply_withdrawal(*withdrawal: WithdrawalAPI*) → *None*

clear_transient_storage() → *None*

Clear the transient storage. Should be done at the start of every transaction

commit(*snapshot: Tuple[Hash32, JournalDBCcheckpoint]*) → *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: SignedTransactionAPI*) → *ComputationAPI*

Execute the given *transaction* with a gas price of 0.

delete_account(*address: Address*) → *None*

Delete the account at the given *address*.

delete_code(*address: Address*) → *None*

Delete the code at *address*.

delete_storage(*address: Address*) → *None*

Delete the storage at *address*

delta_balance(*address*: *Address*, *delta*: *int*) → *None*

Apply delta to the balance at *address*.

classmethod get_account_db_class() → *Type[AccountDatabaseAPI]*

Return the *AccountDatabaseAPI* class that the state class uses.

get_ancestor_hash(*block_number*: *int*) → *Hash32*

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*: *Address*) → *int*

Return the balance for the account at *address*.

get_code(*address*: *Address*) → *bytes*

Return the code at *address*.

get_code_hash(*address*: *Address*) → *Hash32*

Return the hash of the code at *address*.

get_computation(*message*: *MessageAPI*, *transaction_context*: *TransactionContextAPI*) → *ComputationAPI*

Return a computation instance for the given *message* and *transaction_context*

get_gas_price(*transaction*: *SignedTransactionAPI*) → *int*

Return the gas price of the given transaction.

Factor in the current block's base gas price, if appropriate. (See EIP-1559)

get_nonce(*address*: *Address*) → *int*

Return the nonce at *address*.

get_storage(*address*: *Address*, *slot*: *int*, *from_journal*: *bool* = *True*) → *int*

Return the storage at *slot* for *address*.

get_tip(*transaction*: *SignedTransactionAPI*) → *int*

Return the gas price that gets allocated to the miner/validator.

Pre-EIP-1559 that would be the full transaction gas price. After, it would be the tip price (potentially reduced, if the base fee is so high that it surpasses the transaction's maximum gas price after adding the tip).

get_transaction_context(*transaction*: *SignedTransactionAPI*) → *TransactionContextAPI*

Return the *TransactionContextAPI* for the given *transaction*

classmethod get_transaction_context_class() → *Type[TransactionContextAPI]*

Return the *BaseTransactionContext* class that the state class uses.

get_transaction_executor() → *TransactionExecutorAPI*

Return the transaction executor.

get_transient_storage(*address*: *Address*, *slot*: *int*) → *bytes*

Return the transient storage for *address* at slot *slot*.

has_code_or_nonce(*address*: *Address*) → *bool*

Return True if either a nonce or code exists at the given *address*.

increment_nonce(*address*: *Address*) → *None*

Increment the nonce at *address*.

is_address_warm(*address: Address*) → bool

Was the account accessed during this transaction?

See EIP-2929

is_storage_warm(*address: Address, slot: int*) → bool

Was the storage slot accessed during this transaction?

See EIP-2929

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() → Hash32

Create and return the state root.

mark_address_warm(*address: Address*) → None

Mark the account as accessed during this transaction.

See EIP-2929

mark_storage_warm(*address: Address, slot: int*) → None

Mark the storage slot as accessed during this transaction.

See EIP-2929

override_transaction_context(*gas_price: int*) → Iterator[None]

Return a [ContextManager](#) that overwrites the current transaction context, applying the given *gas_price*.

persist() → MetaWitnessAPI

Persist the current state to the database.

revert(*snapshot: Tuple[Hash32, JournalDBCheckpoint]*) → None

Revert the VM to the state at the snapshot

set_balance(*address: Address, balance: int*) → None

Set balance to the balance at *address*.

set_code(*address: Address, code: bytes*) → None

Set code as the new code at *address*.

set_nonce(*address: Address, nonce: int*) → None

Set nonce as the new nonce at *address*.

set_storage(*address: Address, slot: int, value: int*) → None

Write value to the given slot at *address*.

set_transient_storage(*address: Address, slot: int, value: bytes*) → None

Return the transient storage for *address* at slot *slot*.

snapshot() → Tuple[Hash32, JournalDBCheckpoint]

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: Address*) → *None*

Touch the account at the given address.

account_db_class: *Type[AccountDatabaseAPI]* = *None*

property base_fee: *int*

Return the current *base_fee* from the current *execution_context*

Raises a *NotImplementedError* if called in an execution context prior to the London hard fork.

property blob_base_fee: *int*

Return the current *blob_base_fee* from the current *execution_context*

Raises a *NotImplementedError* if called in an execution context prior to the Cancun hard fork.

property block_number: *BlockNumber*

Return the current *block_number* from the current *execution_context*

property coinbase: *Address*

Return the current *coinbase* from the current *execution_context*

computation_class: *Type[ComputationAPI]* = *None*

property difficulty: *int*

Return the current *difficulty* from the current *execution_context*

execution_context: *ExecutionContextAPI*

property gas_limit: *int*

Return the current *gas_limit* from the current *transaction_context*

property logger: *ExtendedDebugLogger*

Return the logger.

property mix_hash: *Hash32*

Return the current *mix_hash* from the current *execution_context*

property state_root: *Hash32*

Return the current *state_root* from the underlying database

property timestamp: *int*

Return the current *timestamp* from the current *execution_context*

transaction_context_class: *Type[TransactionContextAPI]* = *None*

transaction_executor_class: *Type[TransactionExecutorAPI]* = *None*

BaseTransactionExecutor

```
class eth.vm.state.BaseTransactionExecutor(vm_state: StateAPI)
```

BaseTransactionContext

```
class eth.vm.transaction_context.BaseTransactionContext(gas_price: int, origin: Address,
                                                         blob_versioned_hashes:
                                                         Sequence[Hash32] | None = None)
```

get_next_log_counter() → int

Increment and return the log counter.

property blob_versioned_hashes: Sequence[Hash32]

Return the blob versioned hashes of the transaction context.

property gas_price: int

Return the gas price of the transaction context.

property origin: Address

Return the origin of the transaction context.

Forks

Frontier

FrontierVM

```
class eth.vm.forks.frontier.FrontierVM(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI,
                                         chain_context: ChainContextAPI, consensus_context:
                                         ConsensusContextAPI)
```

block_class

alias of FrontierBlock

add_receipt_to_header(old_header: BlockHeaderAPI, receipt: ReceiptAPI) → 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.

classmethod calculate_net_gas_refund(consumed_gas: int, gross_refund: int) → int

static compute_difficulty(parent_header: BlockHeaderAPI, timestamp: int) → int

Computes the difficulty for a frontier block based on the parent block.

configure_header(**header_params: Any) → BlockHeader

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_header_from_parent(parent_header: BlockHeaderAPI, **header_params: Any) → BlockHeader

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

classmethod finalize_gas_used(transaction: SignedTransactionAPI, computation: ComputationAPI) → int

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: BlockHeaderAPI)` → `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

increment_blob_gas_used(*old_header: BlockHeaderAPI, transaction: TransactionFieldsAPI*) → *BlockHeaderAPI*

Update the header by incrementing the `blob_gas_used` for the transaction.

classmethod `make_receipt(base_header: BlockHeaderAPI, transaction: SignedTransactionAPI, computation: ComputationAPI, state: StateAPI)` → *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

validate_transaction_against_header(*base_header: BlockHeaderAPI, transaction: 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

fork: `str` = `'frontier'`

FrontierState

class eth.vm.forks.frontier.state.**FrontierState**(db: *AtomicDatabaseAPI*, execution_context: *ExecutionContextAPI*, state_root: *Hash32*)

account_db_class

alias of *AccountDB*

computation_class

alias of *FrontierComputation*

transaction_context_class

alias of *FrontierTransactionContext*

transaction_executor_class

alias of *FrontierTransactionExecutor*

apply_transaction(transaction: *SignedTransactionAPI*) → *ComputationAPI*

Apply transaction to the vm state

Parameters

transaction – the transaction to apply

Returns

the computation

validate_transaction(transaction: *SignedTransactionAPI*) → *None*

Validate the given transaction.

execution_context: *ExecutionContextAPI*

FrontierComputation

class eth.vm.forks.frontier.computation.**FrontierComputation**(state: *StateAPI*, message: *MessageAPI*, transaction_context: *TransactionContextAPI*)

A class for all execution message computations in the Frontier fork. Inherits from *BaseComputation*

classmethod **apply_create_message**(state: *StateAPI*, message: *MessageAPI*, transaction_context: *TransactionContextAPI*, parent_computation: *ComputationAPI* | *None* = *None*) → *ComputationAPI*

Execute a VM message to create a new contract. This is where the VM-specific create logic exists.

classmethod **apply_message**(state: *StateAPI*, message: *MessageAPI*, transaction_context: *TransactionContextAPI*, parent_computation: *ComputationAPI* | *None* = *None*) → *ComputationAPI*

Execute a VM message. This is where the VM-specific call logic exists.

[illegible]

Homestead

HomesteadVM

```
class eth.vm.forks.homestead.HomesteadVM(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI,
                                          chain_context: ChainContextAPI, consensus_context:
                                          ConsensusContextAPI)

    block_class
        alias of HomesteadBlock

    static compute_difficulty(parent_header: BlockHeaderAPI, timestamp: int) → int
        Computes the difficulty for a homestead block based on the parent block.

    configure_header(**header_params: Any) → BlockHeader
        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_header_from_parent(parent_header: BlockHeaderAPI, **header_params: Any) →
        BlockHeader
        Creates and initializes a new block header from the provided parent_header.

    fork: str = 'homestead'
```

HomesteadState

```
class eth.vm.forks.homestead.state.HomesteadState(db: AtomicDatabaseAPI, execution_context:
                                                  ExecutionContextAPI, state_root: Hash32)

    computation_class
        alias of HomesteadComputation

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

    execution_context: ExecutionContextAPI
```

HomesteadComputation

```
class eth.vm.forks.homestead.computation.HomesteadComputation(state: StateAPI, message:
                                                             MessageAPI, transaction_context:
                                                             TransactionContextAPI)

    A class for all execution message computations in the Frontier fork. Inherits from FrontierComputation

    classmethod apply_create_message(state: StateAPI, message: MessageAPI, transaction_context:
                                    TransactionContextAPI, parent_computation: ComputationAPI |
                                    None = None) → ComputationAPI

        Execute a VM message to create a new contract. This is where the VM-specific create logic exists.
```


[illegible]

TangerineWhistle

TangerineWhistleVM

```
class eth.vm.forks.tangerine_whistle.TangerineWhistleVM(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI, chain_context: ChainContextAPI, consensus_context: ConsensusContextAPI)
```

```
    fork: str = 'tangerine-whistle'
```

```
    support_dao_fork = False
```

TangerineWhistleState

```
class eth.vm.forks.tangerine_whistle.state.TangerineWhistleState(db: AtomicDatabaseAPI, execution_context: ExecutionContextAPI, state_root: Hash32)
```

```
    computation_class
```

```
        alias of TangerineWhistleComputation
```

```
    execution_context: ExecutionContextAPI
```

TangerineWhistleComputation

```
class eth.vm.forks.tangerine_whistle.computation.TangerineWhistleComputation(state: StateAPI, message: MessageAPI, transaction_context: TransactionContextAPI)
```

A class for all execution *message* computations in the *TangerineWhistle* fork. Inherits from *HomesteadComputation*

[illegible]

SpuriousDragon

SpuriousDragonVM

```
class eth.vm.forks.spurious_dragon.SpuriousDragonVM(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI, chain_context: ChainContextAPI, consensus_context: ConsensusContextAPI)
```

block_class

alias of SpuriousDragonBlock

fork: `str` = 'spurious-dragon'

SpuriousDragonState

```
class eth.vm.forks.spurious_dragon.state.SpuriousDragonState(db: AtomicDatabaseAPI, execution_context: ExecutionContextAPI, state_root: Hash32)
```

computation_class

alias of *SpuriousDragonComputation*

transaction_executor_class

alias of SpuriousDragonTransactionExecutor

execution_context: *ExecutionContextAPI*

SpuriousDragonComputation

```
class eth.vm.forks.spurious_dragon.computation.SpuriousDragonComputation(state: StateAPI, message: MessageAPI, transaction_context: TransactionContextAPI)
```

A class for all execution *message* computations in the SpuriousDragon fork. Inherits from *HomesteadComputation*

```
classmethod apply_create_message(state: StateAPI, message: MessageAPI, transaction_context: TransactionContextAPI, parent_computation: ComputationAPI | None = None) → ComputationAPI
```

Execute a VM message to create a new contract. This is where the VM-specific create logic exists.

```
classmethod validate_contract_code(contract_code: bytes) → None
```

```
classmethod validate_create_message(message: MessageAPI) → None
```

Class method for validating a create message.

[illegible]

Byzantium

ByzantiumVM

```
class eth.vm.forks.byzantium.ByzantiumVM(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI,
                                           chain_context: ChainContextAPI, consensus_context:
                                           ConsensusContextAPI)
```

block_class

alias of ByzantiumBlock

```
add_receipt_to_header(old_header: BlockHeaderAPI, receipt: ReceiptAPI) → 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

```
classmethod make_receipt(base_header: BlockHeaderAPI, transaction: SignedTransactionAPI,
                           computation: ComputationAPI, state: StateAPI) → 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: ReceiptAPI) → None
```

Validate the given receipt.

compute_difficulty

<https://github.com/ethereum/EIPs/issues/100>

configure_header

create_header_from_parent

```
fork: str = 'byzantium'
```

get_uncle_reward

ByzantiumState

```
class eth.vm.forks.byzantium.state.ByzantiumState(db: AtomicDatabaseAPI, execution_context:
                                                    ExecutionContextAPI, state_root: Hash32)

    computation_class
        alias of ByzantiumComputation

    execution_context: ExecutionContextAPI
```

ByzantiumComputation

```
class eth.vm.forks.byzantium.computation.ByzantiumComputation(state: StateAPI, message:
                                                                MessageAPI, transaction_context:
                                                                TransactionContextAPI)
```

A class for all execution *message* computations in the Byzantium fork. Inherits from *SpuriousDragonComputation*

[illegible]

Constantinople

ConstantinopleVM

```
class eth.vm.forks.constantinople.ConstantinopleVM(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI, chain_context: ChainContextAPI, consensus_context: ConsensusContextAPI)
```

block_class

alias of 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>

configure_header

create_header_from_parent

```
fork: str = 'constantinople'
```

get_uncle_reward

ConstantinopleState

```
class eth.vm.forks.constantinople.state.ConstantinopleState(db: AtomicDatabaseAPI, execution_context: ExecutionContextAPI, state_root: Hash32)
```

computation_class

alias of *ConstantinopleComputation*

```
execution_context: ExecutionContextAPI
```


ConstantinopleComputation

```
class eth.vm.forks.constantinople.computation.ConstantinopleComputation(state: StateAPI,  
                                                                           message:  
                                                                           MessageAPI,  
                                                                           transaction_context:  
                                                                           TransactionContextAPI)
```

A class for all execution *message* computations in the Constantinople fork. Inherits from *ByzantiumComputation*

[illegible]

Petersburg

PetersburgVM

```
class eth.vm.forks.petersburg.PetersburgVM(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI,
                                             chain_context: ChainContextAPI, consensus_context:
                                             ConsensusContextAPI)
```

block_class

alias of 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>

configure_header

create_header_from_parent

fork: str = 'petersburg'

get_uncle_reward

PetersburgState

```
class eth.vm.forks.petersburg.state.PetersburgState(db: AtomicDatabaseAPI, execution_context:
                                                    ExecutionContextAPI, state_root: Hash32)
```

computation_class

alias of *PetersburgComputation*

execution_context: ExecutionContextAPI

PetersburgComputation

```
class eth.vm.forks.petersburg.computation.PetersburgComputation(state: StateAPI, message:
                                                                MessageAPI,
                                                                transaction_context:
                                                                TransactionContextAPI)
```

A class for all execution *message* computations in the Petersburg fork. Inherits from *ByzantiumComputation*

[illegible]

Istanbul

IstanbulVM

```
class eth.vm.forks.istanbul.IstanbulVM(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI,  
                                         chain_context: ChainContextAPI, consensus_context:  
                                         ConsensusContextAPI)
```

block_class

alias of IstanbulBlock

compute_difficulty

<https://github.com/ethereum/EIPs/issues/100>

configure_header

create_header_from_parent

fork: `str` = 'istanbul'

IstanbulState

```
class eth.vm.forks.istanbul.state.IstanbulState(db: AtomicDatabaseAPI, execution_context:  
                                                ExecutionContextAPI, state_root: Hash32)
```

computation_class

alias of *IstanbulComputation*

execution_context: *ExecutionContextAPI*

IstanbulComputation

```
class eth.vm.forks.istanbul.computation.IstanbulComputation(state: StateAPI, message:  
                                                            MessageAPI, transaction_context:  
                                                            TransactionContextAPI)
```

A class for all execution *message* computations in the Istanbul fork. Inherits from *PetersburgComputation*

[illegible]

Muir Glacier

Submodules

eth.vm.forks.muir_glacier.blocks module

```
class eth.vm.forks.muir_glacier.blocks.MuirGlacierBlock(header: BlockHeaderAPI, transactions:
    Sequence[SignedTransactionAPI] | None =
    None, uncles: Sequence[BlockHeaderAPI]
    | None = None)
```

Bases: IstanbulBlock

transaction_builder

alias of *MuirGlacierTransaction*

property header

property transactions

property uncles

withdrawals: Tuple[WithdrawalAPI, ...]

eth.vm.forks.muir_glacier.computation module

```
class eth.vm.forks.muir_glacier.computation.MuirGlacierComputation(state: StateAPI, message:
    MessageAPI,
    transaction_context:
    TransactionContextAPI)
```

Bases: *IstanbulComputation*

A class for all execution *message* computations in the MuirGlacier fork. Inherits from IstanbulComputation

[illegible]

eth.vm.forks.muir_glacier.headers module

eth.vm.forks.muir_glacier.opcodes module

eth.vm.forks.muir_glacier.state module

```
class eth.vm.forks.muir_glacier.state.MuirGlacierState(db: AtomicDatabaseAPI, execution_context:
                                                    ExecutionContextAPI, state_root: Hash32)
```

Bases: *IstanbulState*

computation_class

alias of *MuirGlacierComputation*

execution_context: *ExecutionContextAPI*

eth.vm.forks.muir_glacier.transactions module

```
class eth.vm.forks.muir_glacier.transactions.MuirGlacierTransaction(*args, **kwargs)
```

Bases: *IstanbulTransaction*

```
classmethod create_unsigned_transaction(*, nonce: int, gas_price: int, gas: int, to: Address, value:
                                         int, data: bytes) → MuirGlacierUnsignedTransaction
```

Create an unsigned transaction.

property data

property gas

property gas_price

Will raise *AttributeError* if get or set on a 1559 transaction.

property nonce

property r

property s

property to

property v

In old transactions, this v field combines the y_parity bit and the chain ID. All new usages should prefer accessing those fields directly. But if you must access the original v, then you can cast to this API first (after checking that type_id is None).

property value

```
class eth.vm.forks.muir_glacier.transactions.MuirGlacierUnsignedTransaction(*args,
                                                                              **kwargs)
```

Bases: *IstanbulUnsignedTransaction*

```
as_signed_transaction(private_key: PrivateKey, chain_id: int | None = None) →  
    MuirGlacierTransaction
```

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

property data

property gas

property gas_price

property nonce

property to

property value

Module contents

```
class eth.vm.forks.muir_glacier.MuirGlacierVM(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI,  
                                              chain_context: ChainContextAPI, consensus_context:  
                                              ConsensusContextAPI)
```

Bases: *IstanbulVM*

block_class

alias of *MuirGlacierBlock*

compute_difficulty

<https://github.com/ethereum/EIPs/issues/100>

configure_header

create_header_from_parent

fork: str = 'muir-glacier'

Berlin

Submodules

eth.vm.forks.berlin.blocks module

```
class eth.vm.forks.berlin.blocks.BerlinBlock(header: BlockHeaderAPI, transactions:  
                                             Sequence[SignedTransactionAPI] | None = None, uncles:  
                                             Sequence[BlockHeaderAPI] | None = None)
```

Bases: *MuirGlacierBlock*

receipt_builder

alias of *BerlinReceiptBuilder*

transaction_builder

alias of *BerlinTransactionBuilder*

property header

property transactions

property uncles

eth.vm.forks.berlin.computation module

```
class eth.vm.forks.berlin.computation.BerlinComputation(state: StateAPI, message: MessageAPI,  
                                                         transaction_context:  
                                                         TransactionContextAPI)
```

Bases: *MuirGlacierComputation*

A class for all execution *message* computations in the Berlin fork. Inherits from *MuirGlacierComputation*

[illegible]

eth.vm.forks.berlin.constants module**eth.vm.forks.berlin.headers module****eth.vm.forks.berlin.logic module**

```
class eth.vm.forks.berlin.logic.CallCodeEIP2929
```

Bases: *LoadFeeByCacheWarmth*, *CallCodeEIP150*

```
class eth.vm.forks.berlin.logic.CallEIP2929
```

Bases: *LoadFeeByCacheWarmth*, *CallByzantium*

```
class eth.vm.forks.berlin.logic.Create2EIP2929
```

Bases: *Create2*

```
generate_contract_address(stack_data: CreateOpcodeStackData, call_data: bytes, computation:
    ComputationAPI) → Address
```

```
class eth.vm.forks.berlin.logic.CreateEIP2929
```

Bases: *CreateByzantium*

```
generate_contract_address(stack_data: CreateOpcodeStackData, call_data: bytes, computation:
    ComputationAPI) → Address
```

```
class eth.vm.forks.berlin.logic.DelegateCallEIP2929
```

Bases: *LoadFeeByCacheWarmth*, *DelegateCallEIP150*

```
class eth.vm.forks.berlin.logic.LoadFeeByCacheWarmth
```

Bases: *object*

```
get_account_load_fee(computation: ComputationAPI, code_address: Address) → int
```

```
class eth.vm.forks.berlin.logic.StaticCallEIP2929
```

Bases: *LoadFeeByCacheWarmth*, *StaticCall*

```
eth.vm.forks.berlin.logic.balance_eip2929(computation: ComputationAPI) → None
```

```
eth.vm.forks.berlin.logic.extcodecopy_eip2929(computation: ComputationAPI) → None
```

```
eth.vm.forks.berlin.logic.extcodehash_eip2929(computation: ComputationAPI) → None
    Return the code hash for a given address. EIP: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1052.md
```

```
eth.vm.forks.berlin.logic.extcodesize_eip2929(computation: ComputationAPI) → None
```

```
eth.vm.forks.berlin.logic.selfdestruct_eip2929(computation: ComputationAPI) → None
```

```
eth.vm.forks.berlin.logic.sload_eip2929(computation: ComputationAPI) → None
```

```
eth.vm.forks.berlin.logic.sstore_eip2929(computation: ComputationAPI = '__no__default__') → int
```

```
eth.vm.forks.berlin.logic.sstore_eip2929_generic(gas_schedule: NetSStoreGasSchedule =
    '__no__default__', computation: ComputationAPI =
    '__no__default__') → int
```

eth.vm.forks.berlin.opcodes module

eth.vm.forks.berlin.receipts module

class eth.vm.forks.berlin.receipts.**BerlinReceiptBuilder**

Bases: ReceiptBuilderAPI

legacy_sedes

alias of *Receipt*

typed_receipt_class

alias of *TypedReceipt*

classmethod **decode**(*encoded: bytes*) → *ReceiptAPI*

This decodes a receipt that is encoded to either a typed receipt, a legacy receipt, or the body of a typed receipt. It assumes that typed receipts are *not* rlp-encoded first.

If dealing with an object that is always rlp encoded, then use this instead:

```
rlp.decode(encoded, sedes=ReceiptBuilderAPI)
```

For example, you may receive a list of receipts via a devp2p request. Each receipt is either a (legacy) rlp list, or a (new-style) bytestring. Even if the receipt is a bytestring, it's wrapped in an rlp bytestring, in that context. New-style receipts will *not* be wrapped in an RLP bytestring in other contexts. They will just be an EIP-2718 type-byte plus payload of concatenated bytes, which cannot be decoded as RLP. This happens for example, when calculating the receipt root hash.

classmethod **deserialize**(*encoded: bytes | List[bytes]*) → *ReceiptAPI*

Extract a receipt from an encoded RLP object.

This method is used by `rlp.decode(..., sedes=ReceiptBuilderAPI)`.

classmethod **serialize**(*obj: ReceiptAPI*) → *bytes | List[bytes]*

Encode a receipt to a series of bytes used by RLP.

In the case of legacy receipt, it will actually be a list of bytes. That doesn't show up here, because `pyrlp` doesn't export type annotations.

This method is used by `rlp.encode(obj)`.

class eth.vm.forks.berlin.receipts.**TypedReceipt**(*type_id: int, proxy_target: ReceiptAPI*)

Bases: *ReceiptAPI*, ReceiptDecoderAPI

copy(**args: Any*, ***kwargs: Any*) → *ReceiptAPI*

Return a copy of the receipt, optionally overwriting any of its properties.

classmethod **decode**(*encoded: bytes*) → *ReceiptAPI*

This decodes a receipt that is encoded to either a typed receipt, a legacy receipt, or the body of a typed receipt. It assumes that typed receipts are *not* rlp-encoded first.

If dealing with an object that is always rlp encoded, then use this instead:

```
rlp.decode(encoded, sedes=ReceiptBuilderAPI)
```

For example, you may receive a list of receipts via a devp2p request. Each receipt is either a (legacy) rlp list, or a (new-style) bytestring. Even if the receipt is a bytestring, it's wrapped in an rlp bytestring, in that context. New-style receipts will *not* be wrapped in an RLP bytestring in other contexts. They will just be an EIP-2718 type-byte plus payload of concatenated bytes, which cannot be decoded as RLP. This happens for example, when calculating the receipt root hash.

```

classmethod deserialize(encoded_unchecked: bytes | List[bytes]) → ReceiptAPI

encode() → bytes
    This encodes a receipt, no matter if it's: a legacy receipt, a typed receipt, or the payload of a typed receipt.
    See more context in decode.

classmethod get_payload_codec(type_id: int) → Type[ReceiptDecoderAPI]

classmethod serialize(obj: TypedReceipt) → bytes | List[bytes]

property bloom: int

property bloom_filter: BloomFilter

codecs = {1: <class 'eth.rlp.receipts.Receipt'>}

property gas_used: int

property logs: Sequence[LogAPI]

rlp_type = <rlp.sedes.binary.Binary object>

property state_root: bytes

type_id: int

```

eth.vm.forks.berlin.state module

```

class eth.vm.forks.berlin.state.BerlinState(db: AtomicDatabaseAPI, execution_context: ExecutionContextAPI, state_root: Hash32)

    Bases: MuirGlacierState

    computation_class
        alias of BerlinComputation

    transaction_executor_class
        alias of BerlinTransactionExecutor

    execution_context: ExecutionContextAPI

class eth.vm.forks.berlin.state.BerlinTransactionExecutor(vm_state: StateAPI)

    Bases: SpuriousDragonTransactionExecutor

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

```

eth.vm.forks.berlin.transactions module

```

class eth.vm.forks.berlin.transactions.AccessListPayloadDecoder

    Bases: TransactionDecoderAPI

    classmethod decode(payload: bytes) → SignedTransactionAPI

        This decodes a transaction that is encoded to either a typed transaction or a legacy transaction, or even the
        payload of one of the transaction types. It assumes that typed transactions are not rlp-encoded first.

        If dealing with an object that is rlp encoded first, then use this instead:

```

```
rlp.decode(encoded, sedes=TransactionBuilderAPI)
```

For example, you may receive a list of transactions via a devp2p request. Each transaction is either a (legacy) rlp list, or a (new-style) bytestring. Even if the transaction is a bytestring, it's wrapped in an rlp bytestring, in that context. New-style transactions will *not* be wrapped in an RLP bytestring in other contexts. They will just be an EIP-2718 type-byte plus payload of concatenated bytes, which cannot be decoded as RLP. An example context for this is calculating the transaction root hash.

```
class eth.vm.forks.berlin.transactions.AccessListTransaction(*args, **kwargs)
```

Bases: `Serializable`, `SignedTransactionMethods`, `SignedTransactionAPI`

check_signature_validity() → `None`

Check if the signature is valid. Raise a `ValidationError` if the signature is invalid.

encode() → `bytes`

This encodes a transaction, no matter if it's: a legacy transaction, a typed transaction, or the payload of a typed transaction. See more context in `decode`.

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.

get_message_for_signing() → `bytes`

Return the bytestring that should be signed in order to create a signed transaction.

get_sender() → `Address`

Get the 20-byte address which sent this transaction.

This can be a slow operation. `transaction.sender` is always preferred.

make_receipt(status: `bytes`, gas_used: `int`, log_entries: `Tuple[Tuple[bytes, Tuple[int, ...], bytes], ...]`) → `ReceiptAPI`

Build a receipt for this transaction.

Transactions have this responsibility because there are different types of transactions, which have different types of receipts. (See access-list transactions, which change the receipt encoding)

Parameters

- **status** – success or failure (used to be the state root after execution)
- **gas_used** – cumulative usage of this transaction and the previous ones in the header
- **log_entries** – logs generated during execution

property access_list

Get addresses to be accessed by a transaction, and their storage slots.

property blob_versioned_hashes: `Hash32`

property chain_id

property data

property gas

property gas_price

Will raise `AttributeError` if get or set on a 1559 transaction.

hash

property `max_fee_per_blob_gas`: `int`

property `max_fee_per_gas`: `int`

Will default to `gas_price` if this is a pre-1559 transaction.

property `max_priority_fee_per_gas`: `int`

Will default to `gas_price` if this is a pre-1559 transaction.

property `nonce`

property `r`

property `s`

property `to`

property `value`

property `y_parity`

The bit used to disambiguate elliptic curve signatures.

The only values this method will return are 0 or 1.

class `eth.vm.forks.berlin.transactions.AccountAccesses(*args, **kwargs)`

Bases: `Serializable`

property `account`

property `storage_keys`

class `eth.vm.forks.berlin.transactions.BerlinLegacyTransaction(*args, **kwargs)`

Bases: `MuirGlacierTransaction`

property `data`

property `gas`

property `gas_price`

Will raise `AttributeError` if get or set on a 1559 transaction.

property `nonce`

property `r`

property `s`

property `to`

property `v`

In old transactions, this `v` field combines the `y_parity` bit and the chain ID. All new usages should prefer accessing those fields directly. But if you must access the original `v`, then you can cast to this API first (after checking that `type_id` is `None`).

property `value`

class `eth.vm.forks.berlin.transactions.BerlinTransactionBuilder`

Bases: `TransactionBuilderInterface`

Responsible for serializing transactions of ambiguous type.

It dispatches to either the legacy transaction type or the new typed transaction, depending on the nature of the encoded/decoded transaction.

legacy_signed

alias of *BerlinLegacyTransaction*

legacy_unsigned

alias of *BerlinUnsignedLegacyTransaction*

typed_transaction

alias of *TypedTransaction*

classmethod **create_unsigned_transaction**(**nonce: int, gas_price: int, gas: int, to: Address, value: int, data: bytes*) → *UnsignedTransactionAPI*

Create an unsigned transaction.

classmethod **decode**(*encoded: bytes*) → *SignedTransactionAPI*

This decodes a transaction that is encoded to either a typed transaction or a legacy transaction, or even the payload of one of the transaction types. It assumes that typed transactions are *not* rlp-encoded first.

If dealing with an object that is rlp encoded first, then use this instead:

```
rlp.decode(encoded, sedes=TransactionBuilderAPI)
```

For example, you may receive a list of transactions via a devp2p request. Each transaction is either a (legacy) rlp list, or a (new-style) bytestring. Even if the transaction is a bytestring, it's wrapped in an rlp bytestring, in that context. New-style transactions will *not* be wrapped in an RLP bytestring in other contexts. They will just be an EIP-2718 type-byte plus payload of concatenated bytes, which cannot be decoded as RLP. An example context for this is calculating the transaction root hash.

classmethod **deserialize**(*encoded: bytes | List[bytes]*) → *SignedTransactionAPI*

Extract a transaction from an encoded RLP object.

This method is used by `rlp.decode(..., sedes=TransactionBuilderAPI)`.

classmethod **new_access_list_transaction**(*chain_id: int, nonce: int, gas_price: int, gas: int, to: Address, value: int, data: bytes, access_list: Sequence[Tuple[Address, Sequence[int]]], y_parity: int, r: int, s: int*) → *TypedTransaction*

classmethod **new_transaction**(*nonce: int, gas_price: int, gas: int, to: Address, value: int, data: bytes, v: int, r: int, s: int*) → *SignedTransactionAPI*

Create a signed transaction.

classmethod **new_unsigned_access_list_transaction**(*chain_id: int, nonce: int, gas_price: int, gas: int, to: Address, value: int, data: bytes, access_list: Sequence[Tuple[Address, Sequence[int]]]*) → *UnsignedAccessListTransaction*

classmethod **serialize**(*obj: SignedTransactionAPI*) → *bytes | List[bytes]*

Encode a transaction to a series of bytes used by RLP.

In the case of legacy transactions, it will actually be a list of bytes. That doesn't show up here, because `pyrlp` doesn't export type annotations.

This method is used by `rlp.encode(obj)`.

class `eth.vm.forks.berlin.transactions.BerlinUnsignedLegacyTransaction(*args, **kwargs)`

Bases: *MuirGlacierUnsignedTransaction*

as_signed_transaction(*private_key*: *PrivateKey*, *chain_id*: *int* | *None* = *None*) → *BerlinLegacyTransaction*

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

property data

property gas

property gas_price

property nonce

property to

property value

class `eth.vm.forks.berlin.transactions.TypedTransaction`(*type_id*: *int*, *proxy_target*: *SignedTransactionAPI*)

Bases: *SignedTransactionMethods*, *SignedTransactionAPI*, *TransactionDecoderAPI*

receipt_builder

alias of *BerlinReceiptBuilder*

check_signature_validity() → *None*

Check if the signature is valid. Raise a *ValidationError* if the signature is invalid.

copy(***overrides*: *Any*) → *TypedTransaction*

Return a copy of the transaction.

classmethod **decode**(*encoded*: *bytes*) → *SignedTransactionAPI*

This decodes a transaction that is encoded to either a typed transaction or a legacy transaction, or even the payload of one of the transaction types. It assumes that typed transactions are *not* rlp-encoded first.

If dealing with an object that is rlp encoded first, then use this instead:

```
rlp.decode(encoded, sedes=TransactionBuilderAPI)
```

For example, you may receive a list of transactions via a devp2p request. Each transaction is either a (legacy) rlp list, or a (new-style) bytestring. Even if the transaction is a bytestring, it's wrapped in an rlp bytestring, in that context. New-style transactions will *not* be wrapped in an RLP bytestring in other contexts. They will just be an EIP-2718 type-byte plus payload of concatenated bytes, which cannot be decoded as RLP. An example context for this is calculating the transaction root hash.

classmethod **deserialize**(*encoded_unchecked*: *bytes* | *List[bytes]*) → *SignedTransactionAPI*

encode() → *bytes*

This encodes a transaction, no matter if it's: a legacy transaction, a typed transaction, or the payload of a typed transaction. See more context in *decode*.

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.

get_message_for_signing() → *bytes*

Return the bytestring that should be signed in order to create a signed transaction.

classmethod **get_payload_codec**(*type_id*: *int*) → *Type[TransactionDecoderAPI]*

get_sender() → Address

Get the 20-byte address which sent this transaction.

This can be a slow operation. `transaction.sender` is always preferred.

make_receipt(*status*: bytes, *gas_used*: int, *log_entries*: Tuple[Tuple[bytes, Tuple[int, ...], bytes], ...]) → *ReceiptAPI*

Build a receipt for this transaction.

Transactions have this responsibility because there are different types of transactions, which have different types of receipts. (See access-list transactions, which change the receipt encoding)

Parameters

- **status** – success or failure (used to be the state root after execution)
- **gas_used** – cumulative usage of this transaction and the previous ones in the header
- **log_entries** – logs generated during execution

classmethod serialize(*obj*: TypedTransaction) → bytes | List[bytes]

property access_list: Sequence[Tuple[Address, Sequence[int]]]

Get addresses to be accessed by a transaction, and their storage slots.

property blob_versioned_hashes: Hash32

property chain_id: int

property data: bytes

decoders: Dict[int, Type[TransactionDecoderAPI]] = {1: <class 'eth.vm.forks.berlin.transactions.AccessListPayloadDecoder'>}

property gas: int

property gas_price: int

Will raise `AttributeError` if get or set on a 1559 transaction.

hash

property max_fee_per_blob_gas: int

property max_fee_per_gas: int

Will default to `gas_price` if this is a pre-1559 transaction.

property max_priority_fee_per_gas: int

Will default to `gas_price` if this is a pre-1559 transaction.

property nonce: int

property r: int

rlp_type = <rlp.sedes.binary.Binary object>

property s: int

property to: Address

property value: int

property y_parity: `int`

The bit used to disambiguate elliptic curve signatures.

The only values this method will return are 0 or 1.

class `eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction(*args, **kwargs)`

Bases: `Serializable`, `UnsignedTransactionAPI`

as_signed_transaction(*private_key*: `PrivateKey`, *chain_id*: `int` | `None` = `None`) → `TypedTransaction`

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

gas_used_by(*computation*: `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.

get_message_for_signing() → `bytes`

validate() → `None`

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

property access_list

Get addresses to be accessed by a transaction, and their storage slots.

property chain_id

property data

property gas

property gas_price

property intrinsic_gas: `int`

Convenience property for the return value of *get_intrinsic_gas*

property max_fee_per_gas: `int`

property max_priority_fee_per_gas: `int`

property nonce

property to

property value

Module contents

class `eth.vm.forks.berlin.BerlinVM(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI, chain_context: ChainContextAPI, consensus_context: ConsensusContextAPI)`

Bases: `MuirGlacierVM`

block_class

alias of `BerlinBlock`

```
compute_difficulty
    https://github.com/ethereum/EIPs/issues/100
configure_header
create_header_from_parent
fork:  str = 'berlin'
```

London

Submodules

eth.vm.forks.london.blocks module

class eth.vm.forks.london.blocks.LondonBackwardsHeader

Bases: BlockHeaderSedesAPI

An rlp sedes class for block headers.

It can serialize and deserialize *both* London and pre-London headers.

classmethod deserialize(encoded: List[bytes]) → BlockHeaderAPI

Extract a header from an encoded RLP object.

This method is used by rlp.decode(..., sedes=TransactionBuilderAPI).

classmethod serialize(obj: BlockHeaderAPI) → List[bytes]

Encode a header to a series of bytes used by RLP.

This method is used by rlp.encode(obj).

class eth.vm.forks.london.blocks.LondonBlock(header: BlockHeaderAPI, transactions:
Sequence[SignedTransactionAPI] | None = None, uncles:
Sequence[BlockHeaderAPI] | None = None)

Bases: BerlinBlock

receipt_builder

alias of LondonReceiptBuilder

transaction_builder

alias of LondonTransactionBuilder

property header

property transactions

property uncles

```

class eth.vm.forks.london.blocks.LondonBlockHeader(difficulty: int, block_number: BlockNumber,
                                                    gas_limit: int, timestamp: int | None = None,
                                                    coinbase: Address =
                                                    b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00',
                                                    parent_hash: Hash32 =
                                                    b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00',
                                                    uncles_hash: Hash32 =
                                                    b'\x1d\xccM\xe8\xde\xc7]z\xab\x85\xb5g\xb6\xcc\xd4\x1a\xd3\x12E',
                                                    state_root: Hash32 =
                                                    b'V\xe8\x1f\x17\x1b\xccU\xa6\xff\x83E\xe6\x92\xcd\xcf8n[H\xe0\x1b',
                                                    transaction_root: Hash32 =
                                                    b'V\xe8\x1f\x17\x1b\xccU\xa6\xff\x83E\xe6\x92\xcd\xcf8n[H\xe0\x1b',
                                                    receipt_root: Hash32 =
                                                    b'V\xe8\x1f\x17\x1b\xccU\xa6\xff\x83E\xe6\x92\xcd\xcf8n[H\xe0\x1b',
                                                    bloom: int = 0, gas_used: int = 0, extra_data:
                                                    bytes = b'', mix_hash: Hash32 =
                                                    b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00',
                                                    nonce: bytes = b'\x00\x00\x00\x00\x00\x00\x00\x00B',
                                                    base_fee_per_gas: int = 0)

```

Bases: `Serializable`, `BlockHeaderAPI`

property base_fee_per_gas

Return the base fee per gas of the block.

Set to None in pre-EIP-1559 (London) header.

property blob_gas_used: int

Return blob gas used.

property block_number

property bloom

property coinbase

property difficulty

property excess_blob_gas: int

Return excess blob gas.

property extra_data

property gas_limit

property gas_used

property hash: Hash32

Return the hash of the block header.

property hex_hash: str

Return the hash as a hex string.

property is_genesis: bool

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

property mining_hash: Hash32

Return the mining hash of the block header.

property `mix_hash`

property `nonce`

property `parent_beacon_block_root`: `Hash32` | `None`

Return the hash of the parent beacon block.

property `parent_hash`

property `receipt_root`

property `state_root`

property `timestamp`

property `transaction_root`

property `uncles_hash`

property `withdrawals_root`: `Hash32` | `None`

Return the withdrawals root of the block.

Set to `None` in pre-Shanghai header.

class `eth.vm.forks.london.blocks.LondonMiningHeader(*args, **kwargs)`

Bases: `Serializable`, *`MiningHeaderAPI`*

property `base_fee_per_gas`

Return the base fee per gas of the block.

Set to `None` in pre-EIP-1559 (London) header.

property `block_number`

property `bloom`

property `coinbase`

property `difficulty`

property `extra_data`

property `gas_limit`

property `gas_used`

property `parent_hash`

property `receipt_root`

property `state_root`

property `timestamp`

property `transaction_root`

property `uncles_hash`

eth.vm.forks.london.computation module

```
class eth.vm.forks.london.computation.LondonComputation(state: StateAPI, message: MessageAPI,  
                                                         transaction_context:  
                                                         TransactionContextAPI)
```

Bases: *BerlinComputation*

A class for all execution *message* computations in the London fork. Inherits from BerlinComputation

```
classmethod validate_contract_code(contract_code: bytes) → None
```

[illegible]

eth.vm.forks.london.constants module**eth.vm.forks.london.headers module****class eth.vm.forks.london.headers.LondonBackwardsHeader**

Bases: `BlockHeaderSedesAPI`

An rlp sedes class for block headers.

It can serialize and deserialize *both* London and pre-London headers.

classmethod `deserialize(encoded: List[bytes]) → BlockHeaderAPI`

Extract a header from an encoded RLP object.

This method is used by `rlp.decode(..., sedes=TransactionBuilderAPI)`.

classmethod `serialize(obj: BlockHeaderAPI) → List[bytes]`

Encode a header to a series of bytes used by RLP.

This method is used by `rlp.encode(obj)`.

`eth.vm.forks.london.headers.calculate_expected_base_fee_per_gas(parent_header: BlockHeaderAPI) → int`

`eth.vm.forks.london.headers.create_london_header_from_parent(difficulty_fn: Callable[[BlockHeaderAPI, int], int] = '__no__default__', parent_header: BlockHeaderAPI | None = '__no__default__', **header_params: Any) → BlockHeaderAPI`

eth.vm.forks.london.opcodes module**eth.vm.forks.london.receipts module****class eth.vm.forks.london.receipts.LondonReceiptBuilder**

Bases: `BerlinReceiptBuilder`

typed_receipt_class

alias of `LondonTypedReceipt`

class `eth.vm.forks.london.receipts.LondonTypedReceipt(type_id: int, proxy_target: ReceiptAPI)`

Bases: `TypedReceipt`

codects: `Dict[int, Type[Receipt]] = {1: <class 'eth.rlp.receipts.Receipt'>, 2: <class 'eth.rlp.receipts.Receipt'>}`

eth.vm.forks.london.state module

class eth.vm.forks.london.state.**LondonState**(*db*: AtomicDatabaseAPI, *execution_context*: ExecutionContextAPI, *state_root*: Hash32)

Bases: *BerlinState*

computation_class

alias of *LondonComputation*

transaction_executor_class

alias of *LondonTransactionExecutor*

get_gas_price(*transaction*: SignedTransactionAPI) → int

Return the gas price of the given transaction.

Factor in the current block's base gas price, if appropriate. (See EIP-1559)

get_tip(*transaction*: SignedTransactionAPI) → int

Return the gas price that gets allocated to the miner/validator.

Pre-EIP-1559 that would be the full transaction gas price. After, it would be the tip price (potentially reduced, if the base fee is so high that it surpasses the transaction's maximum gas price after adding the tip).

get_transaction_context(*transaction*: SignedTransactionAPI) → *TransactionContextAPI*

London-specific transaction context creation, where *gas_price* includes the block base fee

validate_transaction(*transaction*: SignedTransactionAPI) → None

Validate the given transaction.

property base_fee: int

Return the current *base_fee* from the current *execution_context*

Raises a *NotImplementedError* if called in an execution context prior to the London hard fork.

execution_context: *ExecutionContextAPI*

class eth.vm.forks.london.state.**LondonTransactionExecutor**(*vm_state*: StateAPI)

Bases: *BerlinTransactionExecutor*

build_evm_message(*transaction*: SignedTransactionAPI) → *MessageAPI*

Build and return a *MessageAPI* from the given transaction.

classmethod calculate_gas_refund(*computation*: ComputationAPI, *gas_used*: int) → int

eth.vm.forks.london.storage module

eth.vm.forks.london.transactions module

class eth.vm.forks.london.transactions.**DynamicFeePayloadDecoder**

Bases: *TransactionDecoderAPI*

classmethod decode(*payload*: bytes) → *SignedTransactionAPI*

This decodes a transaction that is encoded to either a typed transaction or a legacy transaction, or even the payload of one of the transaction types. It assumes that typed transactions are *not* rlp-encoded first.

If dealing with an object that is rlp encoded first, then use this instead:

```
rlp.decode(encoded, sedes=TransactionBuilderAPI)
```

For example, you may receive a list of transactions via a devp2p request. Each transaction is either a (legacy) rlp list, or a (new-style) bytestring. Even if the transaction is a bytestring, it's wrapped in an rlp bytestring, in that context. New-style transactions will *not* be wrapped in an RLP bytestring in other contexts. They will just be an EIP-2718 type-byte plus payload of concatenated bytes, which cannot be decoded as RLP. An example context for this is calculating the transaction root hash.

```
class eth.vm.forks.london.transactions.DynamicFeeTransaction(*args, **kwargs)
```

Bases: `Serializable`, `SignedTransactionMethods`, [*SignedTransactionAPI*](#)

```
check_signature_validity() → None
```

Check if the signature is valid. Raise a `ValidationError` if the signature is invalid.

```
encode() → bytes
```

This encodes a transaction, no matter if it's: a legacy transaction, a typed transaction, or the payload of a typed transaction. See more context in `decode`.

```
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.

```
get_message_for_signing() → bytes
```

Return the bytestring that should be signed in order to create a signed transaction.

```
get_sender() → Address
```

Get the 20-byte address which sent this transaction.

This can be a slow operation. `transaction.sender` is always preferred.

```
make_receipt(status: bytes, gas_used: int, log_entries: Tuple[Tuple[bytes, Tuple[int, ...], bytes], ...]) →  
ReceiptAPI
```

Build a receipt for this transaction.

Transactions have this responsibility because there are different types of transactions, which have different types of receipts. (See access-list transactions, which change the receipt encoding)

Parameters

- **status** – success or failure (used to be the state root after execution)
- **gas_used** – cumulative usage of this transaction and the previous ones in the header
- **log_entries** – logs generated during execution

```
property access_list
```

Get addresses to be accessed by a transaction, and their storage slots.

```
property blob_versioned_hashes: Hash32
```

```
property chain_id
```

```
property data
```

```
property gas
```

```
property gas_price: None
```

Will raise `AttributeError` if get or set on a 1559 transaction.

```
hash
```

property `max_fee_per_blob_gas`: `int`

property `max_fee_per_gas`

Will default to `gas_price` if this is a pre-1559 transaction.

property `max_priority_fee_per_gas`

Will default to `gas_price` if this is a pre-1559 transaction.

property `nonce`

property `r`

property `s`

property `to`

property `value`

property `y_parity`

The bit used to disambiguate elliptic curve signatures.

The only values this method will return are 0 or 1.

class `eth.vm.forks.london.transactions.LondonLegacyTransaction(*args, **kwargs)`

Bases: [*BerlinLegacyTransaction*](#)

property `data`

property `gas`

property `gas_price`

Will raise [`AttributeError`](#) if get or set on a 1559 transaction.

property `nonce`

property `r`

property `s`

property `to`

property `v`

In old transactions, this `v` field combines the `y_parity` bit and the chain ID. All new usages should prefer accessing those fields directly. But if you must access the original `v`, then you can cast to this API first (after checking that `type_id` is `None`).

property `value`

class `eth.vm.forks.london.transactions.LondonTransactionBuilder`

Bases: [*BerlinTransactionBuilder*](#)

legacy_signed

alias of [*LondonLegacyTransaction*](#)

legacy_unsigned

alias of [*LondonUnsignedLegacyTransaction*](#)

typed_transaction

alias of [*LondonTypedTransaction*](#)

```
classmethod new_dynamic_fee_transaction(chain_id: int, nonce: int, max_priority_fee_per_gas: int,
                                       max_fee_per_gas: int, gas: int, to: Address, value: int,
                                       data: bytes, access_list: Sequence[Tuple[Address,
                                       Sequence[int]]], y_parity: int, r: int, s: int) →
                                       LondonTypedTransaction
```

```
classmethod new_unsigned_dynamic_fee_transaction(chain_id: int, nonce: int,
                                                max_priority_fee_per_gas: int,
                                                max_fee_per_gas: int, gas: int, to: Address,
                                                value: int, data: bytes, access_list:
                                                Sequence[Tuple[Address, Sequence[int]]]) →
                                                UnsignedDynamicFeeTransaction
```

```
class eth.vm.forks.london.transactions.LondonTypedTransaction(type_id: int, proxy_target:
                                                             SignedTransactionAPI)
```

Bases: *TypedTransaction*

receipt_builder

alias of *LondonReceiptBuilder*

```
decoders: Dict[int, Type[TransactionDecoderAPI]] = {1: <class
'eth.vm.forks.berlin.transactions.AccessListPayloadDecoder'>, 2: <class
'eth.vm.forks.london.transactions.DynamicFeePayloadDecoder'>}
```

```
class eth.vm.forks.london.transactions.LondonUnsignedLegacyTransaction(*args, **kwargs)
```

Bases: *BerlinUnsignedLegacyTransaction*

```
as_signed_transaction(private_key: PrivateKey, chain_id: int | None = None) →
LondonLegacyTransaction
```

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

property data

property gas

property gas_price

property nonce

property to

property value

```
class eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction(*args, **kwargs)
```

Bases: *Serializable*, *UnsignedTransactionAPI*

```
as_signed_transaction(private_key: PrivateKey, chain_id: int | None = None) → TypedTransaction
```

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

```
gas_used_by(computation: 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.

get_message_for_signing() → bytes

validate() → None

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

property access_list

Get addresses to be accessed by a transaction, and their storage slots.

property chain_id

property data

property gas

property intrinsic_gas: int

Convenience property for the return value of *get_intrinsic_gas*

property max_fee_per_gas

property max_priority_fee_per_gas

property nonce

property to

property value

eth.vm.forks.london.validation module

eth.vm.forks.london.validation.validate_london_normalized_transaction(state: StateAPI,
transaction:
SignedTransactionAPI)
→ None

Validates a London normalized transaction.

Raise *eth.exceptions.ValidationError* if the sender cannot afford to send this transaction.

Module contents

class eth.vm.forks.london.LondonVM(header: BlockHeaderAPI, chaindb: ChainDatabaseAPI, chain_context:
ChainContextAPI, consensus_context: ConsensusContextAPI)

Bases: BerlinVM

block_class

alias of LondonBlock

classmethod calculate_net_gas_refund(consumed_gas: int, gross_refund: int) → int

classmethod validate_gas(header: BlockHeaderAPI, parent_header: BlockHeaderAPI) → None

compute_difficulty

<https://github.com/ethereum/EIPs/issues/100>

configure_header

create_header_from_parent

fork: str = 'london'

Arrow Glacier

Submodules

eth.vm.forks.arrow_glacier.blocks module

```
class eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlock(header: BlockHeaderAPI, transactions:
    Sequence[SignedTransactionAPI] | None
    = None, uncles:
    Sequence[BlockHeaderAPI] | None =
    None)
```

Bases: *LondonBlock*

transaction_builder

alias of *ArrowGlacierTransactionBuilder*

property header

property transactions

property uncles

```
class eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader(difficulty: int, block_number:
    BlockNumber, gas_limit: int,
    timestamp: int | None = None,
    coinbase: Address =
    b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00',
    parent_hash: Hash32 =
    b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00',
    uncles_hash: Hash32 =
    b'\x1d\xccM\xe8\xde\xc7]z\xab\x85\xb5g\xb6\xcc\xce',
    state_root: Hash32 =
    b'\V\xe8\x1f\x17\x1b\xccU\xa6\xff\x83E\xe6\x92\xcc0',
    transaction_root: Hash32 =
    b'\V\xe8\x1f\x17\x1b\xccU\xa6\xff\x83E\xe6\x92\xcc0',
    receipt_root: Hash32 =
    b'\V\xe8\x1f\x17\x1b\xccU\xa6\xff\x83E\xe6\x92\xcc0',
    bloom: int = 0, gas_used: int =
    0, extra_data: bytes = b'',
    mix_hash: Hash32 =
    b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00',
    nonce: bytes =
    b'\x00\x00\x00\x00\x00\x00\x00\x00B',
    base_fee_per_gas: int = 0)
```

Bases: *LondonBlockHeader*, *ABC*

property base_fee_per_gas

Return the base fee per gas of the block.

Set to None in pre-EIP-1559 (London) header.

property block_number

property bloom

property `coinbase`
property `difficulty`
property `extra_data`
property `gas_limit`
property `gas_used`
property `mix_hash`
property `nonce`
property `parent_hash`
property `receipt_root`
property `state_root`
property `timestamp`
property `transaction_root`
property `uncles_hash`

class `eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader(*args, **kwargs)`

Bases: [LondonMiningHeader](#), [ABC](#)

property `base_fee_per_gas`

Return the base fee per gas of the block.

Set to None in pre-EIP-1559 (London) header.

property `block_number`
property `bloom`
property `coinbase`
property `difficulty`
property `extra_data`
property `gas_limit`
property `gas_used`
property `parent_hash`
property `receipt_root`
property `state_root`
property `timestamp`
property `transaction_root`
property `uncles_hash`

eth.vm.forks.arrow_glacier.computation module

```
class eth.vm.forks.arrow_glacier.computation.ArrowGlacierComputation(state: StateAPI, message:
    MessageAPI,
    transaction_context:
    TransactionContextAPI)
```

Bases: *LondonComputation*

A class for all execution *message* computations in the ArrowGlacier fork. Inherits from *LondonComputation*

eth.vm.forks.arrow_glacier.headers module

```
eth.vm.forks.arrow_glacier.headers.create_arrow_glacier_header_from_parent(difficulty_fn:
    Callable[[BlockHeaderAPI,
    int], int] =
    '__no_default__',
    parent_header:
    BlockHeaderAPI |
    None =
    '__no_default__',
    **header_params:
    Any) →
    BlockHeaderAPI
```

eth.vm.forks.arrow_glacier.state module

```
class eth.vm.forks.arrow_glacier.state.ArrowGlacierState(db: AtomicDatabaseAPI,
    execution_context: ExecutionContextAPI,
    state_root: Hash32)
```

Bases: *LondonState*

computation_class

alias of *ArrowGlacierComputation*

transaction_executor_class

alias of *ArrowGlacierTransactionExecutor*

execution_context: *ExecutionContextAPI*

```
class eth.vm.forks.arrow_glacier.state.ArrowGlacierTransactionExecutor(vm_state: StateAPI)
```

Bases: *LondonTransactionExecutor*

eth.vm.forks.arrow_glacier.transactions module

```
class eth.vm.forks.arrow_glacier.transactions.ArrowGlacierLegacyTransaction(*args,
    **kwargs)
```

Bases: *LondonLegacyTransaction*, *ABC*

property data

property gas

property gas_price

Will raise `AttributeError` if get or set on a 1559 transaction.

property nonce

property r

property s

property to

property v

In old transactions, this v field combines the y_parity bit and the chain ID. All new usages should prefer accessing those fields directly. But if you must access the original v, then you can cast to this API first (after checking that type_id is None).

property value

class eth.vm.forks.arrow_glacier.transactions.**ArrowGlacierTransactionBuilder**

Bases: *LondonTransactionBuilder*

legacy_signed

alias of *ArrowGlacierLegacyTransaction*

legacy_unsigned

alias of *ArrowGlacierUnsignedLegacyTransaction*

class eth.vm.forks.arrow_glacier.transactions.**ArrowGlacierUnsignedLegacyTransaction**(*args,
**kwargs)

Bases: *LondonUnsignedLegacyTransaction*

as_signed_transaction(private_key: PrivateKey, chain_id: int | None = None) →
ArrowGlacierLegacyTransaction

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

property data

property gas

property gas_price

property nonce

property to

property value

Module contents

class eth.vm.forks.arrow_glacier.**ArrowGlacierVM**(header: BlockHeaderAPI, chaindb:
ChainDatabaseAPI, chain_context: ChainContextAPI,
consensus_context: ConsensusContextAPI)

Bases: *LondonVM*

block_class

alias of *ArrowGlacierBlock*

property base_fee_per_gas

Return the base fee per gas of the block.

Set to None in pre-EIP-1559 (London) header.

property block_number

property bloom

property coinbase

property difficulty

property extra_data

property gas_limit

property gas_used

property mix_hash

property nonce

property parent_hash

property receipt_root

property state_root

property timestamp

property transaction_root

property uncles_hash

class eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader(*args, **kwargs)

Bases: [*ArrowGlacierMiningHeader*](#), [*ABC*](#)

property base_fee_per_gas

Return the base fee per gas of the block.

Set to None in pre-EIP-1559 (London) header.

property block_number

property bloom

property coinbase

property difficulty

property extra_data

property gas_limit

property gas_used

property parent_hash

property receipt_root

```
property state_root
property timestamp
property transaction_root
property uncles_hash
```

eth.vm.forks.gray_glacier.computation module

```
class eth.vm.forks.gray_glacier.computation.GrayGlacierComputation(state: StateAPI, message:
                                                                    MessageAPI,
                                                                    transaction_context:
                                                                    TransactionContextAPI)
```

Bases: *ArrowGlacierComputation*

A class for all execution *message* computations in the GrayGlacier fork. Inherits from *ArrowGlacierComputation*

eth.vm.forks.gray_glacier.headers module

```
eth.vm.forks.gray_glacier.headers.create_gray_glacier_header_from_parent(difficulty_fn:
                                                                    Callable[[BlockHeaderAPI,
                                                                    int], int] =
                                                                    '__no__default__',
                                                                    parent_header:
                                                                    BlockHeaderAPI |
                                                                    None =
                                                                    '__no__default__',
                                                                    **header_params:
                                                                    Any) →
                                                                    BlockHeaderAPI
```

eth.vm.forks.gray_glacier.state module

```
class eth.vm.forks.gray_glacier.state.GrayGlacierState(db: AtomicDatabaseAPI, execution_context:
                                                                    ExecutionContextAPI, state_root: Hash32)
```

Bases: *ArrowGlacierState*

computation_class

alias of *GrayGlacierComputation*

transaction_executor_class

alias of *GrayGlacierTransactionExecutor*

execution_context: *ExecutionContextAPI*

```
class eth.vm.forks.gray_glacier.state.GrayGlacierTransactionExecutor(vm_state: StateAPI)
```

Bases: *ArrowGlacierTransactionExecutor*

eth.vm.forks.gray_glacier.transactions module

```
class eth.vm.forks.gray_glacier.transactions.GrayGlacierLegacyTransaction(*args, **kwargs)
```

Bases: [ArrowGlacierLegacyTransaction](#), [ABC](#)

property data

property gas

property gas_price

Will raise [AttributeError](#) if get or set on a 1559 transaction.

property nonce

property r

property s

property to

property v

In old transactions, this v field combines the y_parity bit and the chain ID. All new usages should prefer accessing those fields directly. But if you must access the original v, then you can cast to this API first (after checking that type_id is None).

property value

```
class eth.vm.forks.gray_glacier.transactions.GrayGlacierTransactionBuilder
```

Bases: [ArrowGlacierTransactionBuilder](#)

legacy_signed

alias of [GrayGlacierLegacyTransaction](#)

legacy_unsigned

alias of [GrayGlacierUnsignedLegacyTransaction](#)

```
class eth.vm.forks.gray_glacier.transactions.GrayGlacierUnsignedLegacyTransaction(*args,
                                                                                      **kwargs)
```

Bases: [ArrowGlacierUnsignedLegacyTransaction](#)

as_signed_transaction(*private_key: PrivateKey, chain_id: int | None = None*) → [GrayGlacierLegacyTransaction](#)

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

property data

property gas

property gas_price

property nonce

property to

property value

Module contents

class `eth.vm.forks.gray_glacier.GrayGlacierVM`(*header*: [BlockHeaderAPI](#), *chaindb*: [ChainDatabaseAPI](#), *chain_context*: [ChainContextAPI](#), *consensus_context*: [ConsensusContextAPI](#))

Bases: [ArrowGlacierVM](#)

block_class

alias of [GrayGlacierBlock](#)

compute_difficulty

<https://github.com/ethereum/EIPs/issues/100>

configure_header

create_header_from_parent

fork: `str` = 'gray-glacier'

4.6 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](#).

4.6.1 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, we need to install all dependencies that are needed to run, develop, and test. This is as easy as navigating to the `py-evm` directory and running:

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

4.6.2 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.

4.6.3 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](#).

4.6.4 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()
```

4.6.5 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](#).

4.6.6 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](#).

If possible, the change to the release notes file should be included in the commit that introduces the feature or bugfix.

4.6.7 Releasing

Final test before each release

Before releasing a new version, build and test the package that will be released:

```
git checkout main && 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"`

4.7 Code of Conduct

4.7.1 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.

4.7.2 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

4.7.3 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.

4.7.4 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.

4.7.5 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.

4.7.6 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>

PYTHON MODULE INDEX

e

`eth.exceptions`, 92

`eth.vm.forks.muir_glacier.state`, 141

`eth.vm.forks.muir_glacier.transactions`, 141

V

`eth.vm.forks.arrow_glacier`, 168

`eth.vm.forks.arrow_glacier.blocks`, 165

`eth.vm.forks.arrow_glacier.computation`, 167

`eth.vm.forks.arrow_glacier.headers`, 167

`eth.vm.forks.arrow_glacier.state`, 167

`eth.vm.forks.arrow_glacier.transactions`, 167

`eth.vm.forks.berlin`, 153

`eth.vm.forks.berlin.blocks`, 142

`eth.vm.forks.berlin.computation`, 143

`eth.vm.forks.berlin.constants`, 145

`eth.vm.forks.berlin.headers`, 145

`eth.vm.forks.berlin.logic`, 145

`eth.vm.forks.berlin.opcodes`, 146

`eth.vm.forks.berlin.receipts`, 146

`eth.vm.forks.berlin.state`, 147

`eth.vm.forks.berlin.transactions`, 147

`eth.vm.forks.gray_glacier`, 173

`eth.vm.forks.gray_glacier.blocks`, 169

`eth.vm.forks.gray_glacier.computation`, 171

`eth.vm.forks.gray_glacier.headers`, 171

`eth.vm.forks.gray_glacier.state`, 171

`eth.vm.forks.gray_glacier.transactions`, 172

`eth.vm.forks.london`, 164

`eth.vm.forks.london.blocks`, 154

`eth.vm.forks.london.computation`, 157

`eth.vm.forks.london.constants`, 159

`eth.vm.forks.london.headers`, 159

`eth.vm.forks.london.opcodes`, 159

`eth.vm.forks.london.receipts`, 159

`eth.vm.forks.london.state`, 160

`eth.vm.forks.london.storage`, 160

`eth.vm.forks.london.transactions`, 160

`eth.vm.forks.london.validation`, 164

`eth.vm.forks.muir_glacier`, 142

`eth.vm.forks.muir_glacier.blocks`, 139

`eth.vm.forks.muir_glacier.computation`, 139

`eth.vm.forks.muir_glacier.headers`, 141

`eth.vm.forks.muir_glacier.opcodes`, 141

A

- `access_list` (*eth.abc.BaseTransactionAPI* property), 46
- `access_list` (*eth.rlp.transactions.BaseTransactionMethods* property), 97
- `access_list` (*eth.vm.forks.berlin.transactions.AccessListTransaction* property), 148
- `access_list` (*eth.vm.forks.berlin.transactions.TypedTransaction* property), 152
- `access_list` (*eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction* property), 153
- `access_list` (*eth.vm.forks.london.transactions.DynamicFeeTransaction* property), 161
- `access_list` (*eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction* property), 164
- `AccessListPayloadDecoder` (class in *eth.vm.forks.berlin.transactions*), 147
- `AccessListTransaction` (class in *eth.vm.forks.berlin.transactions*), 148
- `Account` (class in *eth.rlp.accounts*), 94
- `account` (*eth.vm.forks.berlin.transactions.AccountAccesses* property), 149
- `account_db_class` (*eth.abc.StateAPI* attribute), 66
- `account_db_class` (*eth.vm.forks.frontier.state.FrontierState* attribute), 121
- `account_db_class` (*eth.vm.state.BaseState* attribute), 118
- `account_exists()` (*eth.abc.AccountDatabaseAPI* method), 61
- `account_exists()` (*eth.abc.StateAPI* method), 64
- `account_exists()` (*eth.db.account.AccountDB* method), 82
- `account_exists()` (*eth.vm.state.BaseState* method), 115
- `account_has_code_or_nonce()` (*eth.abc.AccountDatabaseAPI* method), 61
- `account_has_code_or_nonce()` (*eth.db.account.AccountDB* method), 82
- `account_is_empty()` (*eth.abc.AccountDatabaseAPI* method), 61
- `account_is_empty()` (*eth.abc.StateAPI* method), 64
- `account_is_empty()` (*eth.db.account.AccountDB* method), 82
- `AccountAccesses` (class in *eth.vm.forks.berlin.transactions*), 149
- `AccountDatabaseAPI` (class in *eth.abc*), 61
- `AccountDB` (class in *eth.db.account*), 82
- `accounts_to_delete` (*eth.abc.ComputationAPI* attribute), 59
- `accounts_to_delete` (*eth.vm.computation.BaseComputation* attribute), 106
- `AccountStorageDatabaseAPI` (class in *eth.abc*), 60
- `AccountStorageDB` (class in *eth.db.storage*), 91
- `add_child_computation()` (*eth.abc.ComputationAPI* method), 57
- `add_child_computation()` (*eth.vm.computation.BaseComputation* method), 104
- `add_log_entry()` (*eth.abc.ComputationAPI* method), 57
- `add_log_entry()` (*eth.vm.computation.BaseComputation* method), 105
- `add_receipt()` (*eth.abc.ChainDatabaseAPI* method), 51
- `add_receipt()` (*eth.db.chain.ChainDB* method), 85
- `add_receipt_to_header()` (*eth.abc.VirtualMachineAPI* method), 68
- `add_receipt_to_header()` (*eth.vm.forks.byzantium.ByzantiumVM* method), 129
- `add_receipt_to_header()` (*eth.vm.forks.frontier.FrontierVM* method), 119
- `add_transaction()` (*eth.abc.ChainDatabaseAPI* method), 51
- `add_transaction()` (*eth.db.chain.ChainDB* method), 85
- `address` (*eth.abc.LogAPI* attribute), 45
- `address` (*eth.rlp.logs.Log* property), 96
- `apply_all_transactions()` (*eth.abc.VirtualMachineAPI* method), 68
- `apply_all_transactions()` (*eth.vm.base.VM* method), 82

`method`), 111
`apply_all_withdrawals()` (*eth.abc.StateAPI* method), 64
`apply_all_withdrawals()` (*eth.abc.VirtualMachineAPI* method), 68
`apply_all_withdrawals()` (*eth.vm.base.VM* method), 111
`apply_all_withdrawals()` (*eth.vm.state.BaseState* method), 115
`apply_child_computation()` (*eth.abc.ComputationAPI* method), 57
`apply_child_computation()` (*eth.vm.computation.BaseComputation* method), 105
`apply_computation()` (*eth.abc.ComputationAPI* class method), 57
`apply_computation()` (*eth.vm.computation.BaseComputation* class method), 105
`apply_create_message()` (*eth.abc.ComputationAPI* class method), 58
`apply_create_message()` (*eth.vm.computation.BaseComputation* class method), 105
`apply_create_message()` (*eth.vm.forks.frontier.computation.FrontierComputation* class method), 121
`apply_create_message()` (*eth.vm.forks.homestead.computation.HomesteadComputation* class method), 123
`apply_create_message()` (*eth.vm.forks.spurious_dragon.computation.SpuriousDragonComputation* class method), 127
`apply_message()` (*eth.abc.ComputationAPI* class method), 58
`apply_message()` (*eth.vm.computation.BaseComputation* class method), 105
`apply_message()` (*eth.vm.forks.frontier.computation.FrontierComputation* class method), 121
`apply_to()` (*eth.db.diff.DBDiff* method), 87
`apply_transaction()` (*eth.abc.MiningChainAPI* method), 76
`apply_transaction()` (*eth.abc.StateAPI* method), 64
`apply_transaction()` (*eth.abc.VirtualMachineAPI* method), 68
`apply_transaction()` (*eth.chains.base.MiningChain* method), 81
`apply_transaction()` (*eth.vm.base.VM* method), 111
`apply_transaction()` (*eth.vm.forks.frontier.state.FrontierState* method), 121
`apply_withdrawal()` (*eth.abc.StateAPI* method), 64
`apply_withdrawal()` (*eth.vm.base.VM* method), 111
`apply_withdrawal()` (*eth.vm.state.BaseState* method), 115
`ArrowGlacierBlock` (class in *eth.vm.forks.arrow_glacier.blocks*), 165
`ArrowGlacierBlockHeader` (class in *eth.vm.forks.arrow_glacier.blocks*), 165
`ArrowGlacierComputation` (class in *eth.vm.forks.arrow_glacier.computation*), 167
`ArrowGlacierLegacyTransaction` (class in *eth.vm.forks.arrow_glacier.transactions*), 167
`ArrowGlacierMiningHeader` (class in *eth.vm.forks.arrow_glacier.blocks*), 166
`ArrowGlacierState` (class in *eth.vm.forks.arrow_glacier.state*), 167
`ArrowGlacierTransactionBuilder` (class in *eth.vm.forks.arrow_glacier.transactions*), 168
`ArrowGlacierTransactionExecutor` (class in *eth.vm.forks.arrow_glacier.state*), 167
`ArrowGlacierUnsignedLegacyTransaction` (class in *eth.vm.forks.arrow_glacier.transactions*), 168
`ArrowGlacierVM` (class in *eth.vm.forks.arrow_glacier*), 168
`as_dict()` (*eth.abc.MiningHeaderAPI* method), 44
`as_dict()` (*eth.abc.SignedTransactionAPI* method), 48
`as_opcode()` (*eth.abc OpcodeAPI* class method), 54
`as_opcode()` (*eth.vm.opcode Opcode* class method), 110
`as_unsigned_transaction()` (*eth.abc.UnsignedTransactionAPI* method), 47
`as_signed_transaction()` (*eth.vm.forks.arrow_glacier.transactions.ArrowGlacierUnsignedLegacyTransaction* method), 168
`as_signed_transaction()` (*eth.vm.forks.berlin.transactions.BerlinUnsignedLegacyTransaction* method), 150
`as_signed_transaction()` (*eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction* method), 153
`as_signed_transaction()` (*eth.vm.forks.gray_glacier.transactions.GrayGlacierUnsignedLegacyTransaction* method), 172
`as_signed_transaction()` (*eth.vm.forks.london.transactions.LondonUnsignedLegacyTransaction* method), 163
`as_signed_transaction()` (*eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction* method), 163
`as_signed_transaction()` (*eth.vm.forks.muir_glacier.transactions.MuirGlacierUnsignedLegacyTransaction* method), 141
`at_block_number()` (in module *eth.tools.builder.chain*), 101

atomic_batch() (*eth.abc.AtomicDatabaseAPI* method), 50

atomic_batch() (*eth.db.atomic.AtomicDB* method), 84

AtomicDatabaseAPI (class in *eth.abc*), 50

AtomicDB (class in *eth.db.atomic*), 84

AtomicDBWriteBatch (class in *eth.db.atomic*), 84

B

balance (*eth.rlp.accounts.Account* property), 94

balance_eip2929() (in module *eth.vm.forks.berlin.logic*), 145

base_fee (*eth.abc.StateAPI* property), 67

base_fee (*eth.vm.forks.london.state.LondonState* property), 160

base_fee (*eth.vm.state.BaseState* property), 118

base_fee_per_gas (*eth.abc.ExecutionContextAPI* property), 57

base_fee_per_gas (*eth.abc.MiningHeaderAPI* property), 44

base_fee_per_gas (*eth.rlp.headers.BlockHeader* property), 95

base_fee_per_gas (*eth.vm.execution_context.ExecutionContext* property), 108

base_fee_per_gas (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader* property), 165

base_fee_per_gas (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader* property), 166

base_fee_per_gas (*eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader* property), 169

base_fee_per_gas (*eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader* property), 170

base_fee_per_gas (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 155

base_fee_per_gas (*eth.vm.forks.london.blocks.LondonMiningHeader* property), 156

BaseBlock (class in *eth.rlp.blocks*), 94

BaseChain (class in *eth.chains.base*), 77

BaseComputation (class in *eth.vm.computation*), 104

BaseDB (class in *eth.db.backends.base*), 84

BaseState (class in *eth.vm.state*), 115

BaseTransaction (class in *eth.rlp.transactions*), 98

BaseTransactionAPI (class in *eth.abc*), 46

BaseTransactionContext (class in *eth.vm.transaction_context*), 119

BaseTransactionExecutor (class in *eth.vm.state*), 118

BaseTransactionFields (class in *eth.rlp.transactions*), 97

BaseTransactionMethods (class in *eth.rlp.transactions*), 97

BaseUnsignedTransaction (class in *eth.rlp.transactions*), 98

BatchDB (class in *eth.db.batch*), 85

beneficiaries (*eth.abc.ComputationAPI* attribute), 59

beneficiaries (*eth.vm.computation.BaseComputation* attribute), 106

BerlinBlock (class in *eth.vm.forks.berlin.blocks*), 142

BerlinComputation (class in *eth.vm.forks.berlin.computation*), 143

BerlinLegacyTransaction (class in *eth.vm.forks.berlin.transactions*), 149

BerlinReceiptBuilder (class in *eth.vm.forks.berlin.receipts*), 146

BerlinState (class in *eth.vm.forks.berlin.state*), 147

BerlinTransactionBuilder (class in *eth.vm.forks.berlin.transactions*), 149

BerlinTransactionExecutor (class in *eth.vm.forks.berlin.state*), 147

BerlinUnsignedLegacyTransaction (class in *eth.vm.forks.berlin.transactions*), 150

BerlinVM (class in *eth.vm.forks.berlin*), 153

blob_base_fee (*eth.abc.StateAPI* property), 67

blob_base_fee (*eth.vm.state.BaseState* property), 118

blob_gas_used (*eth.abc.BlockHeaderAPI* property), 45

blob_gas_used (*eth.rlp.headers.BlockHeader* property), 95

blob_gas_used (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 155

blob_versioned_hashes (*eth.abc.TransactionContextAPI* property), 54

blob_versioned_hashes (*eth.abc.TransactionFieldsAPI* property), 47

blob_versioned_hashes (*eth.vm.forks.berlin.transactions.AccessListTransaction* property), 148

blob_versioned_hashes (*eth.vm.forks.berlin.transactions.TypedTransaction* property), 152

blob_versioned_hashes (*eth.vm.forks.london.transactions.DynamicFeeTransaction* property), 161

blob_versioned_hashes (*eth.vm.transaction_context.BaseTransactionContext* property), 119

block_class (*eth.vm.base.VM* attribute), 113

block_class (*eth.vm.forks.arrow_glacier.ArrowGlacierVM* attribute), 168

block_class (*eth.vm.forks.berlin.BerlinVM* attribute), 153

block_class (*eth.vm.forks.byzantium.ByzantiumVM* attribute), 129

block_class (*eth.vm.forks.constantinople.ConstantinopleVM* attribute), 132

block_class (*eth.vm.forks.frontier.FrontierVM* attribute), 119

block_class (*eth.vm.forks.gray_glacier.GrayGlacierVM* attribute), 170

- [attribute](#)), 173
 - [block_class](#) ([eth.vm.forks.homestead.HomesteadVM attribute](#)), 123
 - [block_class](#) ([eth.vm.forks.istanbul.IstanbulVM attribute](#)), 137
 - [block_class](#) ([eth.vm.forks.london.LondonVM attribute](#)), 164
 - [block_class](#) ([eth.vm.forks.muir_glacier.MuirGlacierVM attribute](#)), 142
 - [block_class](#) ([eth.vm.forks.petersburg.PetersburgVM attribute](#)), 135
 - [block_class](#) ([eth.vm.forks.spurious_dragon.SpuriousDragonVM attribute](#)), 127
 - [block_number](#) ([eth.abc.ExecutionContextAPI property](#)), 57
 - [block_number](#) ([eth.abc.MiningHeaderAPI attribute](#)), 44
 - [block_number](#) ([eth.abc.StateAPI property](#)), 67
 - [block_number](#) ([eth.rlp.headers.BlockHeader property](#)), 95
 - [block_number](#) ([eth.vm.execution_context.ExecutionContextAPI property](#)), 108
 - [block_number](#) ([eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader property](#)), 165
 - [block_number](#) ([eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader property](#)), 166
 - [block_number](#) ([eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader property](#)), 170
 - [block_number](#) ([eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader property](#)), 170
 - [block_number](#) ([eth.vm.forks.london.blocks.LondonBlockHeader property](#)), 155
 - [block_number](#) ([eth.vm.forks.london.blocks.LondonMiningHeader property](#)), 156
 - [block_number](#) ([eth.vm.state.BaseState property](#)), 118
 - [block_preprocessing\(\)](#) ([eth.vm.base.VM class method](#)), 111
 - [BlockAPI](#) (class in [eth.abc](#)), 49
 - [BlockHeader](#) (class in [eth.rlp.headers](#)), 95
 - [BlockHeaderAPI](#) (class in [eth.abc](#)), 45
 - [BlockNotFound](#), 92
 - [bloom](#) ([eth.abc.MiningHeaderAPI attribute](#)), 44
 - [bloom](#) ([eth.abc.ReceiptAPI property](#)), 46
 - [bloom](#) ([eth.rlp.headers.BlockHeader property](#)), 95
 - [bloom](#) ([eth.rlp.receipts.Receipt property](#)), 96
 - [bloom](#) ([eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader property](#)), 165
 - [bloom](#) ([eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader property](#)), 166
 - [bloom](#) ([eth.vm.forks.berlin.receipts.TypedReceipt property](#)), 147
 - [bloom](#) ([eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader property](#)), 170
 - [bloom](#) ([eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader property](#)), 170
 - [bloom](#) ([eth.vm.forks.london.blocks.LondonBlockHeader property](#)), 155
 - [bloom](#) ([eth.vm.forks.london.blocks.LondonMiningHeader property](#)), 156
 - [bloom_filter](#) ([eth.abc.ReceiptAPI property](#)), 46
 - [bloom_filter](#) ([eth.rlp.receipts.Receipt property](#)), 97
 - [bloom_filter](#) ([eth.vm.forks.berlin.receipts.TypedReceipt property](#)), 147
 - [bloomables](#) ([eth.abc.LogAPI property](#)), 45
 - [bloomables](#) ([eth.rlp.logs.Log property](#)), 96
 - [build_block_with_transactions_and_withdrawals\(\)](#) ([eth.abc.ChainAPI method](#)), 73
 - [build_block_with_transactions_and_withdrawals\(\)](#) ([eth.chains.base.Chain method](#)), 78
 - [build_changeset\(\)](#) ([eth.abc.MiningHeaderAPI method](#)), 44
 - [build_computation\(\)](#) ([eth.abc.TransactionExecutorAPI method](#)), 63
 - [build_computation\(\)](#) ([eth.vm.forks.berlin.state.BerlinTransactionExecutorAPI method](#)), 147
 - [build_evm_message\(\)](#) ([eth.abc.TransactionExecutorAPI method](#)), 63
 - [build_evm_message\(\)](#) ([eth.vm.forks.london.state.LondonTransactionExecutorAPI method](#)), 160
 - [build_state\(\)](#) ([eth.abc.VirtualMachineAPI class method](#)), 68
 - [build_state\(\)](#) ([eth.vm.base.VM class method](#)), 111
 - [Builder](#)-in function
 - [eth.vm.opcode.as_opcode\(\)](#), 43
 - [burns_gas](#) ([eth.exceptions.Revert attribute](#)), 93
 - [burns_gas](#) ([eth.exceptions.VMError attribute](#)), 94
 - [ByzantiumComputation](#) (class in [eth.vm.forks.byzantium.computation](#)), 130
 - [ByzantiumState](#) (class in [eth.vm.forks.byzantium.state](#)), 130
 - [ByzantiumVM](#) (class in [eth.vm.forks.byzantium](#)), 129
- ## C
- [CacheDB](#) (class in [eth.db.cache](#)), 85
 - [calc_data_fee\(\)](#) ([eth.abc.TransactionExecutorAPI method](#)), 63
 - [calculate_expected_base_fee_per_gas\(\)](#) (in module [eth.vm.forks.london.headers](#)), 159
 - [calculate_gas_refund\(\)](#) ([eth.vm.forks.london.state.LondonTransactionExecutor class method](#)), 160
 - [calculate_net_gas_refund\(\)](#) ([eth.vm.forks.frontier.FrontierVM class method](#)), 119

- `calculate_net_gas_refund()`
(*eth.vm.forks.london.LondonVM* class method), 164
- `CallCodeEIP2929` (class in *eth.vm.forks.berlin.logic*), 145
- `CallEIP2929` (class in *eth.vm.forks.berlin.logic*), 145
- `CanonicalHeadNotFound`, 92
- `Chain` (class in *eth.chains.base*), 78
- `chain_id` (*eth.abc.ChainAPI* attribute), 76
- `chain_id` (*eth.abc.ExecutionContextAPI* property), 57
- `chain_id` (*eth.abc.HeaderChainAPI* attribute), 73
- `chain_id` (*eth.abc.TransactionFieldsAPI* property), 47
- `chain_id` (*eth.chains.base.BaseChain* attribute), 77
- `chain_id` (*eth.rlp.transactions.BaseTransactionMethods* property), 97
- `chain_id` (*eth.vm.execution_context.ExecutionContext* property), 108
- `chain_id` (*eth.vm.forks.berlin.transactions.AccessListTransaction* attribute), 147
- `chain_id` (*eth.vm.forks.berlin.transactions.TypedTransaction* attribute), 152
- `chain_id` (*eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction* attribute), 153
- `chain_id` (*eth.vm.forks.london.transactions.DynamicFeeTransaction* attribute), 161
- `chain_id` (*eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction* attribute), 164
- `chain_id()` (in module *eth.tools.builder.chain*), 100
- `chain_split()` (in module *eth.tools.builder.chain*), 101
- `ChainAPI` (class in *eth.abc*), 73
- `ChainDatabaseAPI` (class in *eth.abc*), 51
- `ChainDB` (class in *eth.db.chain*), 85
- `chaindb` (*eth.abc.ChainAPI* attribute), 76
- `chaindb` (*eth.abc.VirtualMachineAPI* attribute), 72
- `chaindb` (*eth.chains.base.BaseChain* attribute), 77
- `chaindb` (*eth.vm.base.VM* attribute), 114
- `chaindb_class` (*eth.chains.base.BaseChain* attribute), 77
- `chaindb_class` (*eth.chains.base.Chain* attribute), 78
- `check_signature_validity()`
(*eth.abc.SignedTransactionAPI* method), 48
- `check_signature_validity()`
(*eth.vm.forks.berlin.transactions.AccessListTransaction* method), 148
- `check_signature_validity()`
(*eth.vm.forks.berlin.transactions.TypedTransaction* method), 151
- `check_signature_validity()`
(*eth.vm.forks.london.transactions.DynamicFeeTransaction* method), 161
- `CheckpointsMustBeCanonical`, 92
- `children` (*eth.abc.ComputationAPI* attribute), 59
- `children` (*eth.vm.computation.BaseComputation* attribute), 106
- `clear()` (*eth.db.batch.BatchDB* method), 85
- `clear()` (*eth.db.journal.JournalDB* method), 89
- `clear_transient_storage()` (*eth.abc.StateAPI* method), 64
- `clear_transient_storage()` (*eth.vm.state.BaseState* method), 115
- `cls_logger` (*eth.vm.base.VM* attribute), 114
- `code` (*eth.abc.ComputationAPI* attribute), 59
- `code` (*eth.abc.MessageAPI* attribute), 53
- `code` (*eth.vm.computation.BaseComputation* attribute), 107
- `code` (*eth.vm.message.Message* attribute), 110
- `code_address` (*eth.abc.MessageAPI* property), 53
- `code_address` (*eth.vm.message.Message* property), 110
- `code_hash` (*eth.rlp.accounts.Account* property), 94
- `codecs` (*eth.vm.forks.berlin.receipts.TypedReceipt* attribute), 147
- `codecs` (*eth.vm.forks.london.receipts.LondonTypedReceipt* attribute), 159
- `CodeStream` (class in *eth.vm.code_stream*), 108
- `CodeStreamAPI` (class in *eth.abc*), 56
- `coinbase` (*eth.abc.ExecutionContextAPI* property), 57
- `coinbase` (*eth.abc.MiningHeaderAPI* attribute), 44
- `coinbase` (*eth.abc.StateAPI* property), 67
- `coinbase` (*eth.abc.TransactionFieldsAPI* attribute), 47
- `coinbase` (*eth.abc.headers.BlockHeader* property), 95
- `coinbase` (*eth.vm.execution_context.ExecutionContext* property), 108
- `coinbase` (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader* property), 165
- `coinbase` (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader* property), 166
- `coinbase` (*eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader* property), 170
- `coinbase` (*eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader* property), 170
- `coinbase` (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 155
- `coinbase` (*eth.vm.forks.london.blocks.LondonMiningHeader* property), 156
- `coinbase` (*eth.vm.state.BaseState* property), 118
- `commit` (*eth.db.journal.JournalDB* attribute), 90
- `commit()` (*eth.abc.AccountDatabaseAPI* method), 61
- `commit()` (*eth.abc.AccountStorageDatabaseAPI* method), 60
- `commit()` (*eth.abc.StateAPI* method), 64
- `commit()` (*eth.db.account.AccountDB* method), 82
- `commit()` (*eth.db.batch.BatchDB* method), 85
- `commit()` (*eth.db.storage.AccountStorageDB* method), 91
- `commit()` (*eth.vm.state.BaseState* method), 115
- `commit_to()` (*eth.db.batch.BatchDB* method), 85
- `commit_to()` (*eth.db.storage.StorageLookup* method), 92

`computation_class` (*eth.abc.StateAPI* attribute), 67
`computation_class` (*eth.vm.forks.arrow_glacier.state.ArrowGlacierState* attribute), 123
`computation_class` (*eth.vm.forks.berlin.state.BerlinState* attribute), 167
`computation_class` (*eth.vm.forks.berlin.state.BerlinState* attribute), 147
`computation_class` (*eth.vm.forks.byzantium.state.ByzantiumState* attribute), 130
`computation_class` (*eth.vm.forks.constantinople.state.ConstantinopleState* attribute), 132
`computation_class` (*eth.vm.forks.frontier.state.FrontierState* attribute), 121
`computation_class` (*eth.vm.forks.gray_glacier.state.GrayGlacierState* attribute), 171
`computation_class` (*eth.vm.forks.homestead.state.HomesteadState* attribute), 123
`computation_class` (*eth.vm.forks.istanbul.state.IstanbulState* attribute), 137
`computation_class` (*eth.vm.forks.london.state.LondonState* attribute), 160
`computation_class` (*eth.vm.forks.muir_glacier.state.MuirGlacierState* attribute), 141
`computation_class` (*eth.vm.forks.petersburg.state.PetersburgState* attribute), 135
`computation_class` (*eth.vm.forks.spurious_dragon.state.SpuriousDragonState* attribute), 127
`computation_class` (*eth.vm.forks.tangerine_whistle.state.TangerineWhistleState* attribute), 125
`computation_class` (*eth.vm.state.BaseState* attribute), 118
`ComputationAPI` (class in *eth.abc*), 57
`compute_difficulty` (*eth.vm.forks.arrow_glacier.ArrowGlacierVM* attribute), 168
`compute_difficulty` (*eth.vm.forks.berlin.BerlinVM* attribute), 153
`compute_difficulty` (*eth.vm.forks.byzantium.ByzantiumVM* attribute), 129
`compute_difficulty` (*eth.vm.forks.constantinople.ConstantinopleVM* attribute), 132
`compute_difficulty` (*eth.vm.forks.gray_glacier.GrayGlacierVM* attribute), 173
`compute_difficulty` (*eth.vm.forks.istanbul.IstanbulVM* attribute), 137
`compute_difficulty` (*eth.vm.forks.london.LondonVM* attribute), 164
`compute_difficulty` (*eth.vm.forks.muir_glacier.MuirGlacierVM* attribute), 142
`compute_difficulty` (*eth.vm.forks.petersburg.PetersburgVM* attribute), 135
`compute_difficulty`() (*eth.abc.VirtualMachineAPI* class method), 68
`compute_difficulty`() (*eth.vm.forks.frontier.FrontierVM* static method), 119
`compute_difficulty`() (*eth.vm.forks.homestead.HomesteadVM* static method), 123
`compute_difficulty`() (*eth.abc.VirtualMachineAPI* attribute), 72
`compute_difficulty` (*eth.abc.VirtualMachineAPI* attribute), 111
`compute_difficulty` (*eth.abc.VirtualMachineAPI* attribute), 72
`compute_difficulty` (*eth.abc.ChainAPI* attribute), 76
`compute_difficulty` (*eth.chains.base.BaseChain* attribute), 77
`compute_difficulty` (*eth.chains.base.Chain* attribute), 78
`ConstantinopleComputation` (class in *eth.vm.forks.constantinople.computation*), 133
`ConstantinopleState` (class in *eth.vm.forks.constantinople.state*), 132
`ConstantinopleVM` (class in *eth.vm.forks.constantinople*), 132
`consume_gas`() (*eth.abc.ComputationAPI* method), 58
`consume_gas`() (*eth.abc.GasMeterAPI* method), 53
`consume_gas`() (*eth.vm.computation.BaseComputation* method), 105
`consume_gas`() (*eth.vm.gas_meter.GasMeter* method), 109
`consume_initcode_gas_cost`()

- (*eth.vm.computation.BaseComputation* class method), 105
- ContractCreationCollision, 92
- contracts_created (*eth.abc.ComputationAPI* attribute), 59
- copy() (*eth.abc.BaseTransactionAPI* method), 46
- copy() (*eth.abc.BlockAPI* method), 49
- copy() (*eth.abc.BlockHeaderAPI* method), 45
- copy() (*eth.abc.MemoryAPI* method), 54
- copy() (*eth.abc.ReceiptAPI* method), 46
- copy() (*eth.vm.forks.berlin.receipts.TypedReceipt* method), 146
- copy() (*eth.vm.forks.berlin.transactions.TypedTransaction* method), 151
- copy() (*eth.vm.memory.Memory* method), 109
- copy() (in module *eth.tools.builder.chain*), 101
- costless_execute_transaction() (*eth.abc.StateAPI* method), 64
- costless_execute_transaction() (*eth.vm.state.BaseState* method), 115
- Create2EIP2929 (class in *eth.vm.forks.berlin.logic*), 145
- create_address (*eth.abc.MessageAPI* attribute), 53
- create_address (*eth.vm.message.Message* attribute), 110
- create_arrow_glacier_header_from_parent() (in module *eth.vm.forks.arrow_glacier.headers*), 167
- create_execution_context() (*eth.abc.VirtualMachineAPI* static method), 68
- create_execution_context() (*eth.vm.base.VM* class method), 111
- create_genesis_header() (*eth.abc.VirtualMachineAPI* class method), 69
- create_genesis_header() (*eth.vm.base.VM* class method), 112
- create_gray_glacier_header_from_parent() (in module *eth.vm.forks.gray_glacier.headers*), 171
- create_header_from_parent (*eth.vm.forks.arrow_glacier.ArrowGlacierVM* attribute), 169
- create_header_from_parent (*eth.vm.forks.berlin.BerlinVM* attribute), 154
- create_header_from_parent (*eth.vm.forks.byzantium.ByzantiumVM* attribute), 129
- create_header_from_parent (*eth.vm.forks.constantinople.ConstantinopleVM* attribute), 132
- create_header_from_parent (*eth.vm.forks.gray_glacier.GrayGlacierVM* attribute), 173
- create_header_from_parent (*eth.vm.forks.istanbul.IstanbulVM* attribute), 137
- create_header_from_parent (*eth.vm.forks.london.LondonVM* attribute), 164
- create_header_from_parent (*eth.vm.forks.muir_glacier.MuirGlacierVM* attribute), 142
- create_header_from_parent (*eth.vm.forks.petersburg.PetersburgVM* attribute), 135
- create_header_from_parent() (*eth.abc.ChainAPI* method), 73
- create_header_from_parent() (*eth.abc.VirtualMachineAPI* class method), 69
- create_header_from_parent() (*eth.chains.base.Chain* method), 78
- create_header_from_parent() (*eth.vm.forks.frontier.FrontierVM* static method), 119
- create_header_from_parent() (*eth.vm.forks.homestead.HomesteadVM* static method), 123
- create_london_header_from_parent() (in module *eth.vm.forks.london.headers*), 159
- create_transaction() (*eth.abc.ChainAPI* method), 73
- create_transaction() (*eth.abc.VirtualMachineAPI* method), 69
- create_transaction() (*eth.chains.base.Chain* method), 78
- create_transaction() (*eth.vm.base.VM* method), 112
- create_unsigned_transaction() (*eth.abc.ChainAPI* method), 74
- create_unsigned_transaction() (*eth.abc.VirtualMachineAPI* class method), 69
- create_unsigned_transaction() (*eth.chains.base.Chain* method), 78
- create_unsigned_transaction() (*eth.vm.base.VM* class method), 112
- create_unsigned_transaction() (*eth.vm.forks.berlin.transactions.BerlinTransactionBuilder* class method), 150
- create_unsigned_transaction() (*eth.vm.forks.muir_glacier.transactions.MuirGlacierTransaction* class method), 141
- CreateEIP2929 (class in *eth.vm.forks.berlin.logic*), 145
- ## D
- dao_fork_at() (in module *eth.tools.builder.chain*), 100
- data (*eth.abc.LogAPI* attribute), 45
- data (*eth.abc.MessageAPI* attribute), 53
- data (*eth.abc.TransactionFieldsAPI* property), 47
- data (*eth.abc.UnsignedTransactionAPI* attribute), 47
- data (*eth.rlp.logs.Log* property), 96

- data (*eth.rlp.transactions.BaseTransaction* property), 98
- data (*eth.rlp.transactions.BaseTransactionFields* property), 97
- data (*eth.rlp.transactions.BaseUnsignedTransaction* property), 98
- data (*eth.vm.forks.arrow_glacier.transactions.ArrowGlacierTransaction* property), 167
- data (*eth.vm.forks.arrow_glacier.transactions.ArrowGlacierTransaction* property), 168
- data (*eth.vm.forks.berlin.transactions.AccessListTransaction* property), 148
- data (*eth.vm.forks.berlin.transactions.BerlinLegacyTransaction* property), 149
- data (*eth.vm.forks.berlin.transactions.BerlinUnsignedLegacyTransaction* property), 151
- data (*eth.vm.forks.berlin.transactions.TypedTransaction* property), 152
- data (*eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction* property), 153
- data (*eth.vm.forks.gray_glacier.transactions.GrayGlacierLegacyTransaction* property), 172
- data (*eth.vm.forks.gray_glacier.transactions.GrayGlacierUnsignedLegacyTransaction* property), 172
- data (*eth.vm.forks.london.transactions.DynamicFeeTransaction* property), 161
- data (*eth.vm.forks.london.transactions.LondonLegacyTransaction* property), 162
- data (*eth.vm.forks.london.transactions.LondonUnsignedLegacyTransaction* property), 163
- data (*eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction* property), 164
- data (*eth.vm.forks.muir_glacier.transactions.MuirGlacierTransaction* property), 141
- data (*eth.vm.forks.muir_glacier.transactions.MuirGlacierUnsignedTransaction* property), 142
- data (*eth.vm.message.Message* attribute), 110
- data_as_bytes (*eth.abc.MessageAPI* property), 53
- data_as_bytes (*eth.vm.message.Message* property), 110
- DatabaseAPI (class in *eth.abc*), 49
- db (*eth.abc.ChainDatabaseAPI* attribute), 52
- db (*eth.abc.HeaderDatabaseAPI* attribute), 51
- db (*eth.db.chain.ChainDB* attribute), 87
- db (*eth.db.header.HeaderDB* attribute), 89
- DBDiff (class in *eth.db.diff*), 87
- DBDiffTracker (class in *eth.db.diff*), 88
- decode() (*eth.rlp.receipts.Receipt* class method), 96
- decode() (*eth.rlp.transactions.BaseTransaction* class method), 98
- decode() (*eth.vm.forks.berlin.receipts.BerlinReceiptBuilder* class method), 146
- decode() (*eth.vm.forks.berlin.receipts.TypedReceipt* class method), 146
- decode() (*eth.vm.forks.berlin.transactions.AccessListPayloadBuilder* class method), 147
- decode() (*eth.vm.forks.berlin.transactions.BerlinTransactionBuilder* class method), 150
- decode() (*eth.vm.forks.berlin.transactions.TypedTransaction* class method), 151
- decode() (*eth.vm.forks.london.blocks.LondonBackwardsHeader* class method), 154
- decode() (*eth.vm.forks.london.headers.LondonBackwardsHeader* class method), 147
- decode() (*eth.vm.forks.london.transactions.DynamicFeePayloadDecoder* class method), 160
- decoders (*eth.vm.forks.london.transactions.LondonTypedTransaction* attribute), 163
- DelegateCallEIP2929 (class in *eth.vm.forks.berlin.logic*), 145
- delete() (*eth.abc.AccountStorageDatabaseAPI* method), 60
- delete() (*eth.abc.DatabaseAPI* method), 49
- delete() (*eth.db.backends.base.BaseDB* method), 84
- delete() (*eth.db.storage.AccountStorageDB* method), 91
- delete_account() (*eth.abc.AccountDatabaseAPI* method), 61
- delete_account() (*eth.abc.StateAPI* method), 64
- delete_account() (*eth.db.account.AccountDB* method), 82
- delete_account() (*eth.vm.state.BaseState* method), 115
- delete_code() (*eth.abc.AccountDatabaseAPI* method), 61
- delete_code() (*eth.abc.StateAPI* method), 64
- delete_code() (*eth.db.account.AccountDB* method), 82
- delete_code() (*eth.vm.state.BaseState* method), 115
- delete_storage() (*eth.abc.AccountDatabaseAPI* method), 61
- delete_storage() (*eth.abc.StateAPI* method), 64
- delete_storage() (*eth.db.account.AccountDB* method), 82
- delete_storage() (*eth.vm.state.BaseState* method), 115
- deleted_keys() (*eth.db.diff.DBDiff* method), 87
- delta_balance() (*eth.abc.StateAPI* method), 64
- delta_balance() (*eth.vm.state.BaseState* method), 115
- depth (*eth.abc.MessageAPI* attribute), 53
- depth (*eth.vm.message.Message* attribute), 110
- deserialize() (*eth.vm.forks.berlin.receipts.BerlinReceiptBuilder* class method), 146
- deserialize() (*eth.vm.forks.berlin.receipts.TypedReceipt* class method), 146
- deserialize() (*eth.vm.forks.berlin.transactions.BerlinTransactionBuilder* class method), 150
- deserialize() (*eth.vm.forks.berlin.transactions.TypedTransaction* class method), 151
- deserialize() (*eth.vm.forks.london.blocks.LondonBackwardsHeader* class method), 154
- deserialize() (*eth.vm.forks.london.headers.LondonBackwardsHeader* class method), 147

- class method*), 159
 - `diff()` (*eth.db.batch.BatchDB method*), 85
 - `diff()` (*eth.db.diff.DBDiffTracker method*), 88
 - `diff()` (*eth.db.journal.JournalDB method*), 90
 - `difficulty` (*eth.abc.ExecutionContextAPI property*), 57
 - `difficulty` (*eth.abc.MiningHeaderAPI attribute*), 44
 - `difficulty` (*eth.abc.StateAPI property*), 67
 - `difficulty` (*eth.rlp.headers.BlockHeader property*), 95
 - `difficulty` (*eth.vm.execution_context.ExecutionContext property*), 108
 - `difficulty` (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader property*), 166
 - `difficulty` (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader property*), 166
 - `difficulty` (*eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader property*), 170
 - `difficulty` (*eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader property*), 170
 - `difficulty` (*eth.vm.forks.london.blocks.LondonBlockHeader property*), 155
 - `difficulty` (*eth.vm.forks.london.blocks.LondonMiningHeader property*), 156
 - `difficulty` (*eth.vm.state.BaseState property*), 118
 - `DiffMissingError` (*class in eth.db.diff*), 88
 - `disable_dao_fork()` (*in module eth.tools.builder.chain*), 100
 - `disable_pow_check()` (*in module eth.tools.builder.chain*), 100
 - `discard()` (*eth.abc.AccountDatabaseAPI method*), 61
 - `discard()` (*eth.abc.AccountStorageDatabaseAPI method*), 60
 - `discard()` (*eth.db.account.AccountDB method*), 82
 - `discard()` (*eth.db.journal.JournalDB method*), 90
 - `discard()` (*eth.db.storage.AccountStorageDB method*), 91
 - `dup()` (*eth.abc.StackAPI method*), 55
 - `dup()` (*eth.vm.stack.Stack method*), 114
 - `DynamicFeePayloadDecoder` (*class in eth.vm.forks.london.transactions*), 160
 - `DynamicFeeTransaction` (*class in eth.vm.forks.london.transactions*), 161
- ## E
- `enable_pow_mining()` (*in module eth.tools.builder.chain*), 100
 - `encode()` (*eth.abc.ReceiptAPI method*), 46
 - `encode()` (*eth.abc.SignedTransactionAPI method*), 48
 - `encode()` (*eth.rlp.receipts.Receipt method*), 96
 - `encode()` (*eth.rlp.transactions.BaseTransaction method*), 98
 - `encode()` (*eth.vm.forks.berlin.receipts.TypedReceipt method*), 147
 - `encode()` (*eth.vm.forks.berlin.transactions.AccessListTransaction method*), 148
 - `encode()` (*eth.vm.forks.berlin.transactions.TypedTransaction method*), 151
 - `encode()` (*eth.vm.forks.london.transactions.DynamicFeeTransaction method*), 161
 - `ensure_header()` (*eth.chains.base.Chain method*), 78
 - `erases_return_data` (*eth.exceptions.Revert attribute*), 93
 - `erases_return_data` (*eth.exceptions.VMError attribute*), 94
 - `error` (*eth.abc.ComputationAPI property*), 59
 - `eth.abc.BlockHeader` (*eth.abc.Computation.BaseComputation property*), 107
 - `eth.abc.MiningHeader` (*eth.abc.ChainAPI method*), 74
 - `estimate_gas()` (*eth.chains.base.Chain method*), 78
 - `eth.exceptions` (*module*), 92
 - `eth.vm.forks.arrow_glacier` (*module*), 168
 - `eth.vm.forks.arrow_glacier.blocks` (*module*), 165
 - `eth.vm.forks.arrow_glacier.computation` (*module*), 167
 - `eth.vm.forks.arrow_glacier.headers` (*module*), 167
 - `eth.vm.forks.arrow_glacier.state` (*module*), 167
 - `eth.vm.forks.arrow_glacier.transactions` (*module*), 167
 - `eth.vm.forks.berlin` (*module*), 153
 - `eth.vm.forks.berlin.blocks` (*module*), 142
 - `eth.vm.forks.berlin.computation` (*module*), 143
 - `eth.vm.forks.berlin.constants` (*module*), 145
 - `eth.vm.forks.berlin.headers` (*module*), 145
 - `eth.vm.forks.berlin.logic` (*module*), 145
 - `eth.vm.forks.berlin.opcodes` (*module*), 146
 - `eth.vm.forks.berlin.receipts` (*module*), 146
 - `eth.vm.forks.berlin.state` (*module*), 147
 - `eth.vm.forks.berlin.transactions` (*module*), 147
 - `eth.vm.forks.gray_glacier` (*module*), 173
 - `eth.vm.forks.gray_glacier.blocks` (*module*), 169
 - `eth.vm.forks.gray_glacier.computation` (*module*), 171

`eth.vm.forks.gray_glacier.headers`
module, 171

`eth.vm.forks.gray_glacier.state`
module, 171

`eth.vm.forks.gray_glacier.transactions`
module, 172

`eth.vm.forks.london`
module, 164

`eth.vm.forks.london.blocks`
module, 154

`eth.vm.forks.london.computation`
module, 157

`eth.vm.forks.london.constants`
module, 159

`eth.vm.forks.london.headers`
module, 159

`eth.vm.forks.london.opcodes`
module, 159

`eth.vm.forks.london.receipts`
module, 159

`eth.vm.forks.london.state`
module, 160

`eth.vm.forks.london.storage`
module, 160

`eth.vm.forks.london.transactions`
module, 160

`eth.vm.forks.london.validation`
module, 164

`eth.vm.forks.muir_glacier`
module, 142

`eth.vm.forks.muir_glacier.blocks`
module, 139

`eth.vm.forks.muir_glacier.computation`
module, 139

`eth.vm.forks.muir_glacier.headers`
module, 141

`eth.vm.forks.muir_glacier.opcodes`
module, 141

`eth.vm.forks.muir_glacier.state`
module, 141

`eth.vm.forks.muir_glacier.transactions`
module, 141

`eth.vm.opcode.as_opcode()`
built-in function, 43

`excess_blob_gas` (*eth.abc.BlockHeaderAPI* property), 45

`excess_blob_gas` (*eth.abc.ExecutionContextAPI* property), 57

`excess_blob_gas` (*eth.rlp.headers.BlockHeader* property), 95

`excess_blob_gas` (*eth.vm.execution_context.ExecutionContext* property), 108

`excess_blob_gas` (*eth.vm.forks.london.blocks.LondonBlockHeader* method), 105

`excess_blob_gas` (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 155

`execute_bytecode()` (*eth.abc.VirtualMachineAPI* method), 69

`execute_bytecode()` (*eth.vm.base.VM* method), 112

`execution()` (in module *eth.tools.fixtures.fillers*), 103

`execution_context` (*eth.abc.StateAPI* attribute), 67

`execution_context` (*eth.vm.forks.arrow_glacier.state.ArrowGlacierState* attribute), 167

`execution_context` (*eth.vm.forks.berlin.state.BerlinState* attribute), 147

`execution_context` (*eth.vm.forks.byzantium.state.ByzantiumState* attribute), 130

`execution_context` (*eth.vm.forks.constantinople.state.ConstantinopleState* attribute), 132

`execution_context` (*eth.vm.forks.frontier.state.FrontierState* attribute), 121

`execution_context` (*eth.vm.forks.gray_glacier.state.GrayGlacierState* attribute), 171

`execution_context` (*eth.vm.forks.homestead.state.HomesteadState* attribute), 123

`execution_context` (*eth.vm.forks.istanbul.state.IstanbulState* attribute), 137

`execution_context` (*eth.vm.forks.london.state.LondonState* attribute), 160

`execution_context` (*eth.vm.forks.muir_glacier.state.MuirGlacierState* attribute), 141

`execution_context` (*eth.vm.forks.petersburg.state.PetersburgState* attribute), 135

`execution_context` (*eth.vm.forks.spurious_dragon.state.SpuriousDragonState* attribute), 127

`execution_context` (*eth.vm.forks.tangerine_whistle.state.TangerineWhistleState* attribute), 125

`execution_context` (*eth.vm.state.BaseState* attribute), 118

`ExecutionContext` (class in *eth.vm.execution_context*), 108

`ExecutionContextAPI` (class in *eth.abc*), 57

`exists()` (*eth.abc.ChainDatabaseAPI* method), 51

`exists()` (*eth.abc.DatabaseAPI* method), 49

`exists()` (*eth.db.backends.base.BaseDB* method), 84

`exists()` (*eth.db.chain.ChainDB* method), 85

`expect()` (in module *eth.tools.fixtures.fillers*), 103

`extcodecopy_eip2929()` (in module *eth.vm.forks.berlin.logic*), 145

`extcodehash_eip2929()` (in module *eth.vm.forks.berlin.logic*), 145

`extcodesize_eip2929()` (in module *eth.vm.forks.berlin.logic*), 145

`extend()` (*eth.abc.MemoryAPI* method), 54

`extend()` (*eth.vm.memory.Memory* method), 109

`extend_memory()` (*eth.abc.ComputationAPI* method), 58

`extend_memory()` (*eth.vm.computation.BaseComputation* method), 105

`extra_data` (*eth.abc.MiningHeaderAPI* attribute), 44

[extra_data \(eth.rlp.headers.BlockHeader property\)](#), 95
[extra_data \(eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader property\)](#), 166
[extra_data \(eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader property\)](#), 166
[extra_data \(eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader property\)](#), 170
[extra_data \(eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader property\)](#), 170
[extra_data \(eth.vm.forks.london.blocks.LondonBlockHeader property\)](#), 155
[extra_data \(eth.vm.forks.london.blocks.LondonMiningHeader property\)](#), 156
[extra_data_max_bytes \(eth.abc.VirtualMachineAPI attribute\)](#), 72
[extra_data_max_bytes \(eth.vm.base.VM attribute\)](#), 114

F

[finalize_block\(\) \(eth.abc.VirtualMachineAPI method\)](#), 69
[finalize_block\(\) \(eth.vm.base.VM method\)](#), 112
[finalize_computation\(\) \(eth.abc.TransactionExecutorAPI method\)](#), 63
[finalize_gas_used\(\) \(eth.vm.forks.frontier.FrontierVM class method\)](#), 119
[flatten\(\) \(eth.db.journal.JournalDB method\)](#), 90
[fork \(eth.abc.VirtualMachineAPI attribute\)](#), 72
[fork \(eth.vm.base.VM attribute\)](#), 114
[fork \(eth.vm.forks.arrow_glacier.ArrowGlacierVM attribute\)](#), 169
[fork \(eth.vm.forks.berlin.BerlinVM attribute\)](#), 154
[fork \(eth.vm.forks.byzantium.ByzantiumVM attribute\)](#), 129
[fork \(eth.vm.forks.constantinople.ConstantinopleVM attribute\)](#), 132
[fork \(eth.vm.forks.frontier.FrontierVM attribute\)](#), 120
[fork \(eth.vm.forks.gray_glacier.GrayGlacierVM attribute\)](#), 173
[fork \(eth.vm.forks.homestead.HomesteadVM attribute\)](#), 123
[fork \(eth.vm.forks.istanbul.IstanbulVM attribute\)](#), 137
[fork \(eth.vm.forks.london.LondonVM attribute\)](#), 164
[fork \(eth.vm.forks.muir_glacier.MuirGlacierVM attribute\)](#), 142
[fork \(eth.vm.forks.petersburg.PetersburgVM attribute\)](#), 135
[fork \(eth.vm.forks.spurious_dragon.SpuriousDragonVM attribute\)](#), 127
[fork \(eth.vm.forks.tangerine_whistle.TangerineWhistleVM attribute\)](#), 125
[fork_at\(\) \(in module eth.tools.builder.chain\)](#), 99

[from_genesis\(\) \(eth.abc.ChainAPI class method\)](#), 74
[from_block_headers\(\) \(eth.chains.base.Chain class method\)](#), 78
[from_mining_header\(\) \(eth.abc.ChainAPI class method\)](#), 74
[from_block_headers\(\) \(eth.abc.HeaderChainAPI class method\)](#), 72
[from_genesis_header\(\) \(eth.chains.base.Chain class method\)](#), 78
[from_header\(\) \(eth.abc.BlockAPI class method\)](#), 49
[FrontierComputation \(class in eth.vm.forks.frontier.computation\)](#), 121
[FrontierState \(class in eth.vm.forks.frontier.state\)](#), 121
[FrontierVM \(class in eth.vm.forks.frontier\)](#), 119
[FullStack](#), 92

G

[GapTrackingCorrupted](#), 92
[gas \(eth.abc.MessageAPI attribute\)](#), 53
[gas \(eth.abc.TransactionFieldsAPI property\)](#), 47
[gas \(eth.abc.UnsignedTransactionAPI attribute\)](#), 47
[gas \(eth.rlp.transactions.BaseTransaction property\)](#), 98
[gas \(eth.rlp.transactions.BaseTransactionFields property\)](#), 97
[gas \(eth.rlp.transactions.BaseUnsignedTransaction property\)](#), 98
[gas \(eth.vm.forks.arrow_glacier.transactions.ArrowGlacierLegacyTransaction property\)](#), 167
[gas \(eth.vm.forks.arrow_glacier.transactions.ArrowGlacierUnsignedLegacyTransaction property\)](#), 168
[gas \(eth.vm.forks.berlin.transactions.AccessListTransaction property\)](#), 148
[gas \(eth.vm.forks.berlin.transactions.BerlinLegacyTransaction property\)](#), 149
[gas \(eth.vm.forks.berlin.transactions.BerlinUnsignedLegacyTransaction property\)](#), 151
[gas \(eth.vm.forks.berlin.transactions.TypedTransaction property\)](#), 152
[gas \(eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction property\)](#), 153
[gas \(eth.vm.forks.gray_glacier.transactions.GrayGlacierLegacyTransaction property\)](#), 172
[gas \(eth.vm.forks.gray_glacier.transactions.GrayGlacierUnsignedLegacyTransaction property\)](#), 172
[gas \(eth.vm.forks.london.transactions.DynamicFeeTransaction property\)](#), 161
[gas \(eth.vm.forks.london.transactions.LondonLegacyTransaction property\)](#), 162
[gas \(eth.vm.forks.london.transactions.LondonUnsignedLegacyTransaction property\)](#), 163
[gas \(eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction property\)](#), 164
[gas \(eth.vm.forks.muir_glacier.transactions.MuirGlacierTransaction property\)](#), 141

`gas` (`eth.vm.forks.muir_glacier.transactions.MuirGlacierUnsignedTransaction` property), 161
property), 142
`gas` (`eth.vm.message.Message` attribute), 110
`gas_cost` (`eth.vm.opcode.Opcode` attribute), 110
`gas_estimator` (`eth.chains.base.Chain` attribute), 80
`gas_limit` (`eth.abc.ExecutionContextAPI` property), 57
`gas_limit` (`eth.abc.MiningHeaderAPI` attribute), 44
`gas_limit` (`eth.abc.StateAPI` property), 67
`gas_limit` (`eth.rlp.headers.BlockHeader` property), 95
`gas_limit` (`eth.vm.execution_context.ExecutionContext` property), 108
`gas_limit` (`eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader` property), 166
`gas_limit` (`eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader` property), 166
`gas_limit` (`eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader` property), 170
`gas_limit` (`eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader` property), 170
`gas_limit` (`eth.vm.forks.london.blocks.LondonBlockHeader` property), 155
`gas_limit` (`eth.vm.forks.london.blocks.LondonMiningHeader` property), 156
`gas_limit` (`eth.vm.state.BaseState` property), 118
`gas_price` (`eth.abc.TransactionContextAPI` property), 54
`gas_price` (`eth.abc.TransactionFieldsAPI` property), 47
`gas_price` (`eth.abc.UnsignedTransactionAPI` attribute), 47
`gas_price` (`eth.rlp.transactions.BaseTransaction` property), 98
`gas_price` (`eth.rlp.transactions.BaseTransactionFields` property), 97
`gas_price` (`eth.rlp.transactions.BaseUnsignedTransaction` property), 98
`gas_price` (`eth.vm.forks.arrow_glacier.transactions.ArrowGlacierTransaction` property), 167
`gas_price` (`eth.vm.forks.arrow_glacier.transactions.ArrowGlacierTransaction` property), 168
`gas_price` (`eth.vm.forks.berlin.transactions.AccessListTransaction` property), 148
`gas_price` (`eth.vm.forks.berlin.transactions.BerlinLegacyTransaction` property), 149
`gas_price` (`eth.vm.forks.berlin.transactions.BerlinUnsignedTransaction` property), 151
`gas_price` (`eth.vm.forks.berlin.transactions.TypedTransaction` property), 152
`gas_price` (`eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction` property), 153
`gas_price` (`eth.vm.forks.gray_glacier.transactions.GrayGlacierTransaction` property), 172
`gas_price` (`eth.vm.forks.gray_glacier.transactions.GrayGlacierTransaction` property), 172
`gas_price` (`eth.vm.forks.london.transactions.DynamicFeeTransaction` method), 105
`gas_price` (`eth.vm.forks.london.transactions.LondonLegacyTransaction` property), 162
`gas_price` (`eth.vm.forks.london.transactions.LondonUnsignedLegacyTransaction` property), 163
`gas_price` (`eth.vm.forks.muir_glacier.transactions.MuirGlacierTransaction` property), 141
`gas_price` (`eth.vm.forks.muir_glacier.transactions.MuirGlacierUnsignedTransaction` property), 142
`gas_price` (`eth.vm.transaction_context.BaseTransactionContext` property), 119
`gas_refunded` (`eth.abc.GasMeterAPI` attribute), 53
`gas_refunded` (`eth.vm.gas_meter.GasMeter` attribute), 109
`gas_remaining` (`eth.abc.GasMeterAPI` attribute), 53
`gas_remaining` (`eth.vm.gas_meter.GasMeter` attribute), 109
`gas_used` (`eth.abc.MiningHeaderAPI` attribute), 44
`gas_used` (`eth.abc.ReceiptAPI` property), 46
`gas_used` (`eth.rlp.headers.BlockHeader` property), 95
`gas_used` (`eth.rlp.receipts.Receipt` property), 97
`gas_used` (`eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader` property), 166
`gas_used` (`eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader` property), 166
`gas_used` (`eth.vm.forks.berlin.receipts.TypedReceipt` property), 147
`gas_used` (`eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader` property), 170
`gas_used` (`eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader` property), 170
`gas_used` (`eth.vm.forks.london.blocks.LondonBlockHeader` property), 155
`gas_used` (`eth.vm.forks.london.blocks.LondonMiningHeader` property), 156
`gas_used_by()` (`eth.abc.BaseTransactionAPI` method), 46
`gas_used_by()` (`eth.abc.LegacyTransaction` method), 97
`gas_used_by()` (`eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction` method), 153
`gas_used_by()` (`eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction` method), 163
`GasMeter` (`eth.vm.gas_meter`), 109
`GasMeterAPI` (class in `eth.abc`), 53
`generate_block_from_parent_header_and_coinbase()` (`eth.abc.VirtualMachineAPI` class method), 69
`generate_block_from_parent_header_and_coinbase()` (`eth.vm.base.VM` class method), 112
`generate_transaction_computation()` (`eth.abc.ComputationAPI` method), 58
`generate_transaction_computation()` (`eth.vm.computation.BaseComputation` method), 105

`generate_contract_address()`
(eth.vm.forks.berlin.logic.Create2EIP2929 method), 145
`generate_contract_address()`
(eth.vm.forks.berlin.logic.CreateEIP2929 method), 145
`genesis()` *(in module eth.tools.builder.chain)*, 100
`get()` *(eth.abc.AccountStorageDatabaseAPI method)*, 60
`get()` *(eth.abc.ChainDatabaseAPI method)*, 51
`get()` *(eth.db.chain.ChainDB method)*, 86
`get()` *(eth.db.storage.AccountStorageDB method)*, 91
`get_accessed_slots()`
(eth.abc.AccountStorageDatabaseAPI method), 60
`get_accessed_slots()`
(eth.db.storage.AccountStorageDB method), 91
`get_account_db_class()` *(eth.abc.StateAPI class method)*, 64
`get_account_db_class()` *(eth.vm.state.BaseState class method)*, 116
`get_account_load_fee()`
(eth.vm.forks.berlin.logic.LoadFeeByCacheWarmup method), 145
`get_accounts_for_deletion()`
(eth.abc.ComputationAPI method), 58
`get_accounts_for_deletion()`
(eth.vm.computation.BaseComputation method), 105
`get_ancestor_hash()` *(eth.abc.StateAPI method)*, 64
`get_ancestor_hash()` *(eth.vm.state.BaseState method)*, 116
`get_ancestors()` *(eth.abc.ChainAPI method)*, 74
`get_ancestors()` *(eth.chains.base.Chain method)*, 78
`get_balance()` *(eth.abc.AccountDatabaseAPI method)*, 61
`get_balance()` *(eth.abc.StateAPI method)*, 65
`get_balance()` *(eth.db.account.AccountDB method)*, 82
`get_balance()` *(eth.vm.state.BaseState method)*, 116
`get_block()` *(eth.abc.ChainAPI method)*, 74
`get_block()` *(eth.abc.VirtualMachineAPI method)*, 69
`get_block()` *(eth.chains.base.Chain method)*, 79
`get_block()` *(eth.vm.base.VM method)*, 112
`get_block_by_hash()` *(eth.abc.ChainAPI method)*, 74
`get_block_by_hash()` *(eth.chains.base.Chain method)*, 79
`get_block_by_header()` *(eth.abc.ChainAPI method)*, 74
`get_block_by_header()` *(eth.chains.base.Chain method)*, 79
`get_block_class()` *(eth.abc.VirtualMachineAPI class method)*, 69
`get_block_class()` *(eth.vm.base.VM class method)*, 112
`get_block_header_by_hash()` *(eth.abc.ChainAPI method)*, 74
`get_block_header_by_hash()` *(eth.abc.HeaderChainAPI method)*, 72
`get_block_header_by_hash()` *(eth.abc.HeaderDatabaseAPI method)*, 50
`get_block_header_by_hash()` *(eth.chains.base.Chain method)*, 79
`get_block_header_by_hash()` *(eth.db.header.HeaderDB method)*, 88
`get_block_reward()` *(eth.abc.VirtualMachineAPI static method)*, 69
`get_block_reward()` *(eth.vm.forks.byzantium.ByzantiumVM static method)*, 129
`get_block_reward()` *(eth.vm.forks.constantinople.ConstantinopleVM static method)*, 132
`get_block_reward()` *(eth.vm.forks.frontier.FrontierVM static method)*, 119
`get_block_reward()` *(eth.vm.forks.petersburg.PetersburgVM static method)*, 135
`get_block_transaction_hashes()`
(eth.abc.ChainDatabaseAPI method), 51
`get_block_transaction_hashes()`
(eth.db.chain.ChainDB method), 86
`get_block_transactions()`
(eth.abc.ChainDatabaseAPI method), 51
`get_block_transactions()` *(eth.db.chain.ChainDB method)*, 86
`get_block_uncles()` *(eth.abc.ChainDatabaseAPI method)*, 51
`get_block_uncles()` *(eth.db.chain.ChainDB method)*, 86
`get_block_withdrawals()`
(eth.abc.ChainDatabaseAPI method), 51
`get_block_withdrawals()` *(eth.db.chain.ChainDB method)*, 86
`get_canonical_block_by_number()`
(eth.abc.ChainAPI method), 74
`get_canonical_block_by_number()`
(eth.chains.base.Chain method), 79
`get_canonical_block_hash()` *(eth.abc.ChainAPI method)*, 74
`get_canonical_block_hash()`
(eth.abc.HeaderChainAPI method), 72
`get_canonical_block_hash()`
(eth.abc.HeaderDatabaseAPI method), 50
`get_canonical_block_hash()` *(eth.chains.base.Chain method)*, 79
`get_canonical_block_hash()`
(eth.db.header.HeaderDB method), 88
`get_canonical_block_header_by_number()`
(eth.abc.ChainAPI method), 74
`get_canonical_block_header_by_number()`
(eth.abc.HeaderChainAPI method), 72
`get_canonical_block_header_by_number()`

[\(eth.abc.HeaderDatabaseAPI method\)](#), 50

[get_canonical_block_header_by_number\(\)](#)
[\(eth.chains.base.Chain method\)](#), 79

[get_canonical_block_header_by_number\(\)](#)
[\(eth.db.header.HeaderDB method\)](#), 88

[get_canonical_head\(\)](#) [\(eth.abc.ChainAPI method\)](#), 75

[get_canonical_head\(\)](#) [\(eth.abc.HeaderChainAPI method\)](#), 72

[get_canonical_head\(\)](#) [\(eth.abc.HeaderDatabaseAPI method\)](#), 50

[get_canonical_head\(\)](#) [\(eth.chains.base.Chain method\)](#), 79

[get_canonical_head\(\)](#) [\(eth.db.header.HeaderDB method\)](#), 88

[get_canonical_transaction\(\)](#) [\(eth.abc.ChainAPI method\)](#), 75

[get_canonical_transaction\(\)](#)
[\(eth.chains.base.Chain method\)](#), 79

[get_canonical_transaction_by_index\(\)](#)
[\(eth.abc.ChainAPI method\)](#), 75

[get_canonical_transaction_by_index\(\)](#)
[\(eth.chains.base.Chain method\)](#), 79

[get_canonical_transaction_index\(\)](#)
[\(eth.abc.ChainAPI method\)](#), 75

[get_canonical_transaction_index\(\)](#)
[\(eth.chains.base.Chain method\)](#), 79

[get_chain_gaps\(\)](#) [\(eth.db.chain.ChainDB method\)](#), 86

[get_chaindb_class\(\)](#) [\(eth.abc.ChainAPI class method\)](#), 75

[get_chaindb_class\(\)](#) [\(eth.chains.base.Chain class method\)](#), 79

[get_changed_root\(\)](#) [\(eth.abc.AccountStorageDatabaseAPI method\)](#), 60

[get_changed_root\(\)](#) [\(eth.db.storage.AccountStorageDB method\)](#), 91

[get_changed_root\(\)](#) [\(eth.db.storage.StorageLookup method\)](#), 92

[get_code\(\)](#) [\(eth.abc.AccountDatabaseAPI method\)](#), 61

[get_code\(\)](#) [\(eth.abc.StateAPI method\)](#), 65

[get_code\(\)](#) [\(eth.db.account.AccountDB method\)](#), 82

[get_code\(\)](#) [\(eth.vm.state.BaseState method\)](#), 116

[get_code_hash\(\)](#) [\(eth.abc.AccountDatabaseAPI method\)](#), 61

[get_code_hash\(\)](#) [\(eth.abc.StateAPI method\)](#), 65

[get_code_hash\(\)](#) [\(eth.db.account.AccountDB method\)](#), 82

[get_code_hash\(\)](#) [\(eth.vm.state.BaseState method\)](#), 116

[get_computation\(\)](#) [\(eth.abc.StateAPI method\)](#), 65

[get_computation\(\)](#) [\(eth.vm.state.BaseState method\)](#), 116

[get_gas_meter\(\)](#) [\(eth.abc.ComputationAPI method\)](#), 58

[get_gas_meter\(\)](#) [\(eth.vm.computation.BaseComputation method\)](#), 105

[get_gas_price\(\)](#) [\(eth.abc.StateAPI method\)](#), 65

[get_gas_price\(\)](#) [\(eth.vm.forks.london.state.LondonState method\)](#), 160

[get_gas_price\(\)](#) [\(eth.vm.state.BaseState method\)](#), 116

[get_gas_refund\(\)](#) [\(eth.abc.ComputationAPI method\)](#), 58

[get_gas_refund\(\)](#) [\(eth.vm.computation.BaseComputation method\)](#), 105

[get_gas_remaining\(\)](#) [\(eth.abc.ComputationAPI method\)](#), 58

[get_gas_remaining\(\)](#)
[\(eth.vm.computation.BaseComputation method\)](#), 105

[get_gas_used\(\)](#) [\(eth.abc.ComputationAPI method\)](#), 58

[get_gas_used\(\)](#) [\(eth.vm.computation.BaseComputation method\)](#), 105

[get_header\(\)](#) [\(eth.abc.VirtualMachineAPI method\)](#), 70

[get_header\(\)](#) [\(eth.vm.base.VM method\)](#), 112

[get_header_chain_gaps\(\)](#)
[\(eth.abc.HeaderDatabaseAPI method\)](#), 50

[get_header_chain_gaps\(\)](#) [\(eth.db.header.HeaderDB method\)](#), 88

[get_headerdb_class\(\)](#) [\(eth.abc.HeaderChainAPI class method\)](#), 72

[get_intrinsic_gas\(\)](#) [\(eth.abc.BaseTransactionAPI method\)](#), 46

[get_intrinsic_gas\(\)](#)
[\(eth.vm.forks.berlin.transactions.AccessListTransaction method\)](#), 148

[get_intrinsic_gas\(\)](#)
[\(eth.vm.forks.berlin.transactions.TypedTransaction method\)](#), 151

[get_intrinsic_gas\(\)](#)
[\(eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction method\)](#), 153

[get_intrinsic_gas\(\)](#)
[\(eth.vm.forks.london.transactions.DynamicFeeTransaction method\)](#), 161

[get_intrinsic_gas\(\)](#)
[\(eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction method\)](#), 163

[get_log_entries\(\)](#) [\(eth.abc.ComputationAPI method\)](#), 58

[get_log_entries\(\)](#) [\(eth.vm.computation.BaseComputation method\)](#), 105

[get_message_for_signing\(\)](#)
[\(eth.abc.SignedTransactionAPI method\)](#), 48

[get_message_for_signing\(\)](#)
[\(eth.vm.forks.berlin.transactions.AccessListTransaction method\)](#), 148

[get_message_for_signing\(\)](#)
[\(eth.vm.forks.berlin.transactions.TypedTransaction method\)](#), 151

- method*), 151
- `get_message_for_signing()` (*eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction* *method*), 153
- `get_message_for_signing()` (*eth.vm.forks.london.transactions.DynamicFeeTransaction* *method*), 161
- `get_message_for_signing()` (*eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction* *method*), 163
- `get_nephew_reward()` (*eth.abc.VirtualMachineAPI class method*), 70
- `get_nephew_reward()` (*eth.vm.forks.frontier.FrontierVM class method*), 120
- `get_next_log_counter()` (*eth.abc.TransactionContextAPI method*), 54
- `get_next_log_counter()` (*eth.vm.transaction_context.BaseTransactionContextAPI method*), 119
- `get_nonce()` (*eth.abc.AccountDatabaseAPI method*), 61
- `get_nonce()` (*eth.abc.StateAPI method*), 65
- `get_nonce()` (*eth.db.account.AccountDB method*), 82
- `get_nonce()` (*eth.vm.state.BaseState method*), 116
- `get_opcode_fn()` (*eth.abc.ComputationAPI method*), 58
- `get_opcode_fn()` (*eth.vm.computation.BaseComputation method*), 106
- `get_payload_codec()` (*eth.vm.forks.berlin.receipts.TypedReceipt class method*), 147
- `get_payload_codec()` (*eth.vm.forks.berlin.transactions.TypedTransaction class method*), 151
- `get_precompiles()` (*eth.abc.ComputationAPI class method*), 58
- `get_precompiles()` (*eth.vm.computation.BaseComputation class method*), 106
- `get_prev_hashes()` (*eth.abc.VirtualMachineAPI class method*), 70
- `get_raw_log_entries()` (*eth.abc.ComputationAPI method*), 58
- `get_raw_log_entries()` (*eth.vm.computation.BaseComputation method*), 106
- `get_receipt_builder()` (*eth.abc.BlockAPI class method*), 49
- `get_receipt_builder()` (*eth.abc.VirtualMachineAPI class method*), 70
- `get_receipt_builder()` (*eth.vm.base.VM class method*), 112
- `get_receipt_by_index()` (*eth.abc.ChainDatabaseAPI method*), 51
- `get_receipt_by_index()` (*eth.db.chain.ChainDB method*), 86
- `get_receipts()` (*eth.abc.BlockAPI method*), 49
- `get_receipts()` (*eth.abc.ChainDatabaseAPI method*), 51
- `get_receipts()` (*eth.db.chain.ChainDB method*), 86
- `get_score()` (*eth.abc.ChainAPI method*), 75
- `get_score()` (*eth.abc.HeaderDatabaseAPI method*), 50
- `get_score()` (*eth.chains.base.Chain method*), 80
- `get_score()` (*eth.db.header.HeaderDB method*), 89
- `get_self_destruct_beneficiaries()` (*eth.abc.ComputationAPI method*), 58
- `get_self_destruct_beneficiaries()` (*eth.vm.computation.BaseComputation method*), 106
- `get_sender()` (*eth.abc.SignedTransactionAPI method*), 48
- `get_sender()` (*eth.vm.forks.berlin.transactions.AccessListTransaction method*), 148
- `get_sender()` (*eth.vm.forks.berlin.transactions.TypedTransaction method*), 151
- `get_sender()` (*eth.vm.forks.london.transactions.DynamicFeeTransaction method*), 161
- `get_state_class()` (*eth.abc.VirtualMachineAPI class method*), 70
- `get_state_class()` (*eth.vm.base.VM class method*), 112
- `get_storage()` (*eth.abc.AccountDatabaseAPI method*), 61
- `get_storage()` (*eth.abc.StateAPI method*), 65
- `get_storage()` (*eth.db.account.AccountDB method*), 82
- `get_storage()` (*eth.vm.state.BaseState method*), 116
- `get_tip()` (*eth.abc.StateAPI method*), 65
- `get_tip()` (*eth.vm.forks.london.state.LondonState method*), 160
- `get_tip()` (*eth.vm.state.BaseState method*), 116
- `get_transaction_builder()` (*eth.abc.BlockAPI class method*), 49
- `get_transaction_builder()` (*eth.abc.VirtualMachineAPI class method*), 70
- `get_transaction_builder()` (*eth.rlp.blocks.BaseBlock class method*), 94
- `get_transaction_builder()` (*eth.vm.base.VM class method*), 112
- `get_transaction_by_index()` (*eth.abc.ChainDatabaseAPI method*), 51
- `get_transaction_by_index()` (*eth.db.chain.ChainDB method*), 86
- `get_transaction_context()` (*eth.abc.StateAPI method*), 65
- `get_transaction_context()` (*eth.vm.forks.london.state.LondonState method*), 160

`get_transaction_context()` (*eth.vm.state.BaseState* method), 116

`get_transaction_context_class()` (*eth.abc.StateAPI* class method), 65

`get_transaction_context_class()` (*eth.vm.state.BaseState* class method), 116

`get_transaction_executor()` (*eth.abc.StateAPI* method), 65

`get_transaction_executor()` (*eth.vm.state.BaseState* method), 116

`get_transaction_index()` (*eth.abc.ChainDatabaseAPI* method), 52

`get_transaction_index()` (*eth.db.chain.ChainDB* method), 86

`get_transaction_receipt()` (*eth.abc.ChainAPI* method), 75

`get_transaction_receipt()` (*eth.chains.base.Chain* method), 80

`get_transaction_receipt_by_index()` (*eth.abc.ChainAPI* method), 75

`get_transaction_receipt_by_index()` (*eth.chains.base.Chain* method), 80

`get_transaction_result()` (*eth.abc.ChainAPI* method), 75

`get_transaction_result()` (*eth.chains.base.Chain* method), 80

`get_transient_storage()` (*eth.abc.StateAPI* method), 65

`get_transient_storage()` (*eth.vm.state.BaseState* method), 116

`get_uncle_reward()` (*eth.vm.forks.byzantium.ByzantiumVM* attribute), 129

`get_uncle_reward()` (*eth.vm.forks.constantinople.ConstantinopleVM* attribute), 132

`get_uncle_reward()` (*eth.vm.forks.petersburg.PetersburgVM* attribute), 135

`get_uncle_reward()` (*eth.abc.VirtualMachineAPI* static method), 70

`get_uncle_reward()` (*eth.vm.forks.frontier.FrontierVM* static method), 120

`get_vm()` (*eth.abc.ChainAPI* method), 75

`get_vm()` (*eth.chains.base.Chain* method), 80

`get_vm()` (*eth.chains.base.MiningChain* method), 81

`get_vm_class()` (*eth.abc.ChainAPI* class method), 75

`get_vm_class()` (*eth.chains.base.BaseChain* class method), 77

`get_vm_class_for_block_number()` (*eth.abc.ChainAPI* class method), 75

`get_vm_class_for_block_number()` (*eth.chains.base.BaseChain* class method), 77

`GrayGlacierBlock` (class in *eth.vm.forks.gray_glacier.blocks*), 169

`GrayGlacierBlockHeader` (class in *eth.vm.forks.gray_glacier.blocks*), 169

`GrayGlacierComputation` (class in *eth.vm.forks.gray_glacier.computation*), 171

`GrayGlacierLegacyTransaction` (class in *eth.vm.forks.gray_glacier.transactions*), 172

`GrayGlacierMiningHeader` (class in *eth.vm.forks.gray_glacier.blocks*), 170

`GrayGlacierState` (class in *eth.vm.forks.gray_glacier.state*), 171

`GrayGlacierTransactionBuilder` (class in *eth.vm.forks.gray_glacier.transactions*), 172

`GrayGlacierTransactionExecutor` (class in *eth.vm.forks.gray_glacier.state*), 171

`GrayGlacierUnsignedLegacyTransaction` (class in *eth.vm.forks.gray_glacier.transactions*), 172

`GrayGlacierVM` (class in *eth.vm.forks.gray_glacier*), 173

H

`Halt`, 92

`has_changed_root` (*eth.abc.AccountStorageDatabaseAPI* property), 61

`has_changed_root` (*eth.db.storage.AccountStorageDB* property), 91

`has_changed_root` (*eth.db.storage.StorageLookup* property), 92

`has_checkpoint()` (*eth.db.journal.JournalDB* method), 90

`has_clear()` (*eth.db.journal.JournalDB* method), 90

`has_code_or_nonce()` (*eth.abc.StateAPI* method), 65

`has_code_or_nonce()` (*eth.vm.state.BaseState* method), 116

`has_root()` (*eth.abc.AccountDatabaseAPI* method), 62

`has_root()` (*eth.db.account.AccountDB* method), 82

`hash` (*eth.abc.BlockAPI* property), 49

`hash` (*eth.abc.MiningHeaderAPI* property), 44

`hash` (*eth.abc.TransactionFieldsAPI* property), 47

`hash` (*eth.rlp.headers.BlockHeader* property), 95

`hash` (*eth.rlp.transactions.BaseTransactionFields* property), 97

`hash` (*eth.vm.forks.berlin.transactions.AccessListTransaction* attribute), 148

`hash` (*eth.vm.forks.berlin.transactions.TypedTransaction* attribute), 152

`hash` (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 155

`hash` (*eth.vm.forks.london.transactions.DynamicFeeTransaction* attribute), 161

`header` (*eth.abc.BlockAPI* attribute), 49

`header` (*eth.abc.HeaderChainAPI* attribute), 73

`header` (*eth.abc.MiningChainAPI* attribute), 77

`header` (*eth.chains.base.MiningChain* attribute), 81

`header` (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlock* property), 165

header (*eth.vm.forks.berlin.blocks.BerlinBlock* property), 142
 header (*eth.vm.forks.gray_glacier.blocks.GrayGlacierBlock* property), 169
 header (*eth.vm.forks.london.blocks.LondonBlock* property), 154
 header (*eth.vm.forks.muir_glacier.blocks.MuirGlacierBlock* property), 139
 header_exists() (*eth.abc.HeaderChainAPI* method), 72
 header_exists() (*eth.abc.HeaderDatabaseAPI* method), 50
 header_exists() (*eth.db.header.HeaderDB* method), 89
 HeaderChainAPI (class in *eth.abc*), 72
 HeaderDatabaseAPI (class in *eth.abc*), 50
 HeaderDB (class in *eth.db.header*), 88
 HeaderNotFound, 92
 hex_hash (*eth.abc.MiningHeaderAPI* property), 44
 hex_hash (*eth.rlp.headers.BlockHeader* property), 95
 hex_hash (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 155
 HomesteadComputation (class in *eth.vm.forks.homestead.computation*), 123
 HomesteadState (class in *eth.vm.forks.homestead.state*), 123
 HomesteadVM (class in *eth.vm.forks.homestead*), 123
 |
 import_block() (*eth.abc.ChainAPI* method), 75
 import_block() (*eth.abc.VirtualMachineAPI* method), 70
 import_block() (*eth.chains.base.Chain* method), 80
 import_block() (*eth.chains.base.MiningChain* method), 81
 import_block() (*eth.vm.base.VM* method), 112
 import_block() (in module *eth.tools.builder.chain*), 101
 import_blocks() (in module *eth.tools.builder.chain*), 101
 import_header() (*eth.abc.HeaderChainAPI* method), 73
 in_costless_state() (*eth.abc.VirtualMachineAPI* method), 70
 in_costless_state() (*eth.vm.base.VM* method), 112
 IncorrectContractCreationAddress, 92
 increment_blob_gas_used() (*eth.abc.VirtualMachineAPI* method), 70
 increment_blob_gas_used() (*eth.vm.forks.frontier.FrontierVM* method), 120
 increment_nonce() (*eth.abc.AccountDatabaseAPI* method), 62
 increment_nonce() (*eth.abc.StateAPI* method), 65
 increment_nonce() (*eth.db.account.AccountDB* method), 82
 increment_nonce() (*eth.vm.state.BaseState* method), 116
 InsufficientFunds, 93
 InsufficientStack, 93
 intrinsic_gas (*eth.abc.BaseTransactionAPI* property), 46
 intrinsic_gas (*eth.rlp.transactions.BaseTransactionMethods* property), 97
 intrinsic_gas (*eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction* property), 153
 intrinsic_gas (*eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction* property), 164
 invalid_positions (*eth.vm.code_stream.CodeStream* attribute), 108
 InvalidInstruction, 93
 InvalidJumpDestination, 93
 is_address_warm() (*eth.abc.AccountDatabaseAPI* method), 62
 is_address_warm() (*eth.abc.StateAPI* method), 65
 is_address_warm() (*eth.db.account.AccountDB* method), 82
 is_address_warm() (*eth.vm.state.BaseState* method), 116
 is_create (*eth.abc.MessageAPI* property), 53
 is_create (*eth.vm.message.Message* property), 110
 is_deleted (*eth.db.diff.DiffMissingError* property), 88
 is_error (*eth.abc.ComputationAPI* property), 59
 is_error (*eth.vm.computation.BaseComputation* property), 107
 is_genesis (*eth.abc.BlockAPI* property), 49
 is_genesis (*eth.abc.MiningHeaderAPI* property), 44
 is_genesis (*eth.rlp.blocks.BaseBlock* property), 94
 is_genesis (*eth.rlp.headers.BlockHeader* property), 95
 is_genesis (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 155
 is_origin_computation (*eth.abc.ComputationAPI* property), 59
 is_origin_computation (*eth.vm.computation.BaseComputation* property), 107
 is_signature_valid (*eth.abc.SignedTransactionAPI* property), 48
 is_static (*eth.abc.MessageAPI* attribute), 53
 is_static (*eth.vm.message.Message* attribute), 110
 is_storage_warm() (*eth.abc.AccountDatabaseAPI* method), 62
 is_storage_warm() (*eth.abc.StateAPI* method), 65
 is_storage_warm() (*eth.db.account.AccountDB* method), 83
 is_storage_warm() (*eth.vm.state.BaseState* method), 117
 is_success (*eth.abc.ComputationAPI* property), 59

- `is_success` (*eth.vm.computation.BaseComputation* property), 107
- `is_valid_opcode()` (*eth.abc.CodeStreamAPI* method), 56
- `is_valid_opcode()` (*eth.vm.code_stream.CodeStream* method), 108
- `IstanbulComputation` (class in *eth.vm.forks.istanbul.computation*), 137
- `IstanbulState` (class in *eth.vm.forks.istanbul.state*), 137
- `IstanbulVM` (class in *eth.vm.forks.istanbul*), 137
- ## J
- `join()` (*eth.db.diff.DBDiff* class method), 87
- `JournalDB` (class in *eth.db.journal*), 89

K

`KeyAccessLoggerDB` (class in *eth.db.accesslog*), 81

`keys_read` (*eth.db.accesslog.KeyAccessLoggerDB* property), 81

`kv_store` (*eth.db.backends.memory.MemoryDB* attribute), 85

L

`legacy_sedes` (*eth.vm.forks.berlin.receipts.BerlinReceiptBuilder* attribute), 146

`legacy_signed` (*eth.vm.forks.arrow_glacier.transactions.ArrowGlacierTransactionBuilder* attribute), 168

`legacy_signed` (*eth.vm.forks.berlin.transactions.BerlinTransactionBuilder* attribute), 149

`legacy_signed` (*eth.vm.forks.gray_glacier.transactions.GrayGlacierTransactionBuilder* attribute), 172

`legacy_signed` (*eth.vm.forks.london.transactions.LondonTransactionBuilder* attribute), 162

`legacy_unsigned` (*eth.vm.forks.arrow_glacier.transactions.ArrowGlacierTransactionBuilder* attribute), 168

`legacy_unsigned` (*eth.vm.forks.berlin.transactions.BerlinTransactionBuilder* attribute), 150

`legacy_unsigned` (*eth.vm.forks.gray_glacier.transactions.GrayGlacierTransactionBuilder* attribute), 172

`legacy_unsigned` (*eth.vm.forks.london.transactions.LondonTransactionBuilder* attribute), 162

`LoadFeeByCacheWarmth` (class in *eth.vm.forks.berlin.logic*), 145

`lock_changes()` (*eth.abc.AccountDatabaseAPI* method), 62

`lock_changes()` (*eth.abc.AccountStorageDatabaseAPI* method), 60

`lock_changes()` (*eth.abc.StateAPI* method), 66

`lock_changes()` (*eth.db.account.AccountDB* method), 83

`lock_changes()` (*eth.db.storage.AccountStorageDB* method), 91

`lock_changes()` (*eth.vm.state.BaseState* method), 117

`Log` (class in *eth.rlp.logs*), 96

`LogAPI` (class in *eth.abc*), 45

`logger` (*eth.abc.ComputationAPI* attribute), 59

`logger` (*eth.abc.StateAPI* property), 67

`logger` (*eth.chains.base.Chain* attribute), 80

`logger` (*eth.db.accesslog.KeyAccessLoggerDB* attribute), 81

`logger` (*eth.db.account.AccountDB* attribute), 84

`logger` (*eth.db.atomic.AtomicDB* attribute), 84

`logger` (*eth.db.atomic.AtomicDBWriteBatch* attribute), 84

`logger` (*eth.db.batch.BatchDB* attribute), 85

`logger` (*eth.db.storage.AccountStorageDB* attribute), 91

`logger` (*eth.db.storage.StorageLookup* attribute), 92

`logger` (*eth.vm.base.VM* property), 114

`logger` (*eth.vm.code_stream.CodeStream* attribute), 108

`logger` (*eth.vm.computation.BaseComputation* attribute), 107

`logger` (*eth.vm.gas_meter.GasMeter* attribute), 109

`logger` (*eth.vm.memory.Memory* attribute), 109

`logger` (*eth.vm.message.Message* attribute), 110

`logger` (*eth.vm.opcode.Opcode* property), 110

`logger` (*eth.vm.stack.Stack* attribute), 115

`logger` (*eth.vm.state.BaseState* property), 118

`logs` (*eth.abc.ReceiptAPI* property), 46

`logs` (*eth.rlp.receipts.Receipt* property), 97

`logs` (*eth.vm.forks.berlin.receipts.TypedReceipt* property), 147

`LondonBackwardsHeader` (class in *eth.vm.forks.london.blocks*), 154

`LondonBackwardsHeader` (class in *eth.vm.forks.london.headers*), 159

`LondonBlock` (class in *eth.vm.forks.london.blocks*), 154

`LondonBlockHeader` (class in *eth.vm.forks.london.blocks*), 154

`LondonComputation` (class in *eth.vm.forks.london.computation*), 157

`LondonLegacyTransaction` (class in *eth.vm.forks.london.transactions*), 162

`LondonMiningHeader` (class in *eth.vm.forks.london.blocks*), 156

`LondonReceiptBuilder` (class in *eth.vm.forks.london.receipts*), 159

`LondonState` (class in *eth.vm.forks.london.state*), 160

`LondonTransactionBuilder` (class in *eth.vm.forks.london.transactions*), 162

`LondonTransactionExecutor` (class in *eth.vm.forks.london.state*), 160

`LondonTypedReceipt` (class in *eth.vm.forks.london.receipts*), 159

`LondonTypedTransaction` (class in *eth.vm.forks.london.transactions*), 163

`LondonUnsignedLegacyTransaction` (class in *eth.vm.forks.london.transactions*), 163

LondonVM (class in *eth.vm.forks.london*), 164

M

make_block_hash_to_score_lookup_key()
(*eth.db.schema.SchemaV1* static method), 90

make_block_number_to_hash_lookup_key()
(*eth.db.schema.SchemaV1* static method), 90

make_canonical_head_hash_lookup_key()
(*eth.db.schema.SchemaV1* static method), 90

make_chain_gaps_lookup_key()
(*eth.db.schema.SchemaV1* static method), 90

make_checkpoint_headers_key()
(*eth.db.schema.SchemaV1* static method), 90

make_header_chain_gaps_lookup_key()
(*eth.db.schema.SchemaV1* static method), 90

make_receipt() (*eth.abc.SignedTransactionAPI* method), 48

make_receipt() (*eth.abc.VirtualMachineAPI* method), 70

make_receipt() (*eth.vm.forks.berlin.transactions.AccessListTransaction* method), 148

make_receipt() (*eth.vm.forks.berlin.transactions.TypedTransaction* method), 152

make_receipt() (*eth.vm.forks.byzantium.ByzantiumVM* class method), 129

make_receipt() (*eth.vm.forks.frontier.FrontierVM* class method), 120

make_receipt() (*eth.vm.forks.london.transactions.DynamicFeeTransaction* method), 161

make_state_root() (*eth.abc.AccountDatabaseAPI* method), 62

make_state_root() (*eth.abc.StateAPI* method), 66

make_state_root() (*eth.db.account.AccountDB* method), 83

make_state_root() (*eth.vm.state.BaseState* method), 117

make_storage_root()
(*eth.abc.AccountStorageDatabaseAPI* method), 60

make_storage_root()
(*eth.db.storage.AccountStorageDB* method), 91

make_transaction_hash_to_block_lookup_key()
(*eth.db.schema.SchemaV1* static method), 91

make_withdrawal_hash_to_block_lookup_key()
(*eth.db.schema.SchemaV1* static method), 91

mark_address_warm() (*eth.abc.AccountDatabaseAPI* method), 62

mark_address_warm() (*eth.abc.StateAPI* method), 66

mark_address_warm() (*eth.db.account.AccountDB* method), 83

mark_address_warm() (*eth.vm.state.BaseState* method), 117

mark_storage_warm() (*eth.abc.AccountDatabaseAPI* method), 62

mark_storage_warm() (*eth.abc.StateAPI* method), 66

mark_storage_warm() (*eth.db.account.AccountDB* method), 83

mark_storage_warm() (*eth.vm.state.BaseState* method), 117

max_fee_per_blob_gas (*eth.abc.TransactionFieldsAPI* property), 47

max_fee_per_blob_gas
(*eth.vm.forks.berlin.transactions.AccessListTransaction* property), 148

max_fee_per_blob_gas
(*eth.vm.forks.berlin.transactions.TypedTransaction* property), 152

max_fee_per_blob_gas
(*eth.vm.forks.london.transactions.DynamicFeeTransaction* property), 161

max_fee_per_gas (*eth.abc.TransactionFieldsAPI* property), 47

max_fee_per_gas (*eth.vm.forks.berlin.transactions.AccessListTransaction* property), 149

max_fee_per_gas (*eth.vm.forks.berlin.transactions.TypedTransaction* property), 152

max_fee_per_gas (*eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction* property), 153

max_fee_per_gas (*eth.vm.forks.london.transactions.DynamicFeeTransaction* property), 162

max_fee_per_gas (*eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction* property), 164

max_priority_fee_per_gas
(*eth.abc.TransactionFieldsAPI* property), 47

max_priority_fee_per_gas
(*eth.vm.forks.berlin.transactions.AccessListTransaction* property), 149

max_priority_fee_per_gas
(*eth.vm.forks.berlin.transactions.TypedTransaction* property), 152

max_priority_fee_per_gas
(*eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction* property), 153

max_priority_fee_per_gas
(*eth.vm.forks.london.transactions.DynamicFeeTransaction* property), 162

max_priority_fee_per_gas
(*eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction* property), 164

Memory (class in *eth.vm.memory*), 109

memory_copy() (*eth.abc.ComputationAPI* method), 59

`memory_copy()` (*eth.vm.computation.BaseComputation method*), 106

`memory_read_bytes()` (*eth.abc.ComputationAPI method*), 59

`memory_read_bytes()` (*eth.vm.computation.BaseComputation method*), 106

`memory_write()` (*eth.abc.ComputationAPI method*), 59

`memory_write()` (*eth.vm.computation.BaseComputation method*), 106

`MemoryAPI` (class in *eth.abc*), 54

`MemoryDB` (class in *eth.db.backends.memory*), 85

`Message` (class in *eth.vm.message*), 110

`MessageAPI` (class in *eth.abc*), 53

`mine_all()` (*eth.abc.MiningChainAPI method*), 76

`mine_all()` (*eth.chains.base.MiningChain method*), 81

`mine_block()` (*eth.abc.MiningChainAPI method*), 77

`mine_block()` (*eth.abc.VirtualMachineAPI method*), 71

`mine_block()` (*eth.chains.base.MiningChain method*), 81

`mine_block()` (*eth.vm.base.VM method*), 113

`mine_block()` (in module *eth.tools.builder.chain*), 101

`mine_block_extended()` (*eth.abc.MiningChainAPI method*), 77

`mine_block_extended()` (*eth.chains.base.MiningChain method*), 81

`mine_blocks()` (in module *eth.tools.builder.chain*), 101

`mining_hash` (*eth.abc.MiningHeaderAPI property*), 44

`mining_hash` (*eth.rlp.headers.BlockHeader property*), 95

`mining_hash` (*eth.vm.forks.london.blocks.LondonBlockHeader property*), 155

`MiningChain` (class in *eth.chains.base*), 81

`MiningChainAPI` (class in *eth.abc*), 76

`MiningHeaderAPI` (class in *eth.abc*), 44

`missing_state_root` (*eth.exceptions.StateRootNotFound property*), 93

`mix_hash` (*eth.abc.BlockHeaderAPI attribute*), 45

`mix_hash` (*eth.abc.ExecutionContextAPI property*), 57

`mix_hash` (*eth.abc.StateAPI property*), 67

`mix_hash` (*eth.rlp.headers.BlockHeader property*), 95

`mix_hash` (*eth.vm.execution_context.ExecutionContext property*), 108

`mix_hash` (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader property*), 166

`mix_hash` (*eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader property*), 170

`mix_hash` (*eth.vm.forks.london.blocks.LondonBlockHeader property*), 155

`mix_hash` (*eth.vm.state.BaseState property*), 118

`mnemonic` (*eth.abc.OpCodeAPI attribute*), 54

`mnemonic` (*eth.vm.opcode.OpCode attribute*), 110

module

- `eth.exceptions`, 92
- `eth.vm.forks.arrow_glacier`, 168
- `eth.vm.forks.arrow_glacier.blocks`, 165
- `eth.vm.forks.arrow_glacier.computation`, 167
- `eth.vm.forks.arrow_glacier.headers`, 167
- `eth.vm.forks.arrow_glacier.state`, 167
- `eth.vm.forks.arrow_glacier.transactions`, 167
- `eth.vm.forks.berlin`, 153
- `eth.vm.forks.berlin.blocks`, 142
- `eth.vm.forks.berlin.computation`, 143
- `eth.vm.forks.berlin.constants`, 145
- `eth.vm.forks.berlin.headers`, 145
- `eth.vm.forks.berlin.logic`, 145
- `eth.vm.forks.berlin.opcodes`, 146
- `eth.vm.forks.berlin.receipts`, 146
- `eth.vm.forks.berlin.state`, 147
- `eth.vm.forks.berlin.transactions`, 147
- `eth.vm.forks.gray_glacier`, 173
- `eth.vm.forks.gray_glacier.blocks`, 169
- `eth.vm.forks.gray_glacier.computation`, 171
- `eth.vm.forks.gray_glacier.headers`, 171
- `eth.vm.forks.gray_glacier.state`, 171
- `eth.vm.forks.gray_glacier.transactions`, 172
- `eth.vm.forks.london`, 164
- `eth.vm.forks.london.blocks`, 154
- `eth.vm.forks.london.computation`, 157
- `eth.vm.forks.london.constants`, 159
- `eth.vm.forks.london.headers`, 159
- `eth.vm.forks.london.opcodes`, 159
- `eth.vm.forks.london.receipts`, 159
- `eth.vm.forks.london.state`, 160
- `eth.vm.forks.london.storage`, 160
- `eth.vm.forks.london.transactions`, 160
- `eth.vm.forks.london.validation`, 164
- `eth.vm.forks.muir_glacier`, 142
- `eth.vm.forks.muir_glacier.blocks`, 139
- `eth.vm.forks.muir_glacier.computation`, 139
- `eth.vm.forks.muir_glacier.headers`, 141
- `eth.vm.forks.muir_glacier.opcodes`, 141
- `eth.vm.forks.muir_glacier.state`, 141
- `eth.vm.forks.muir_glacier.transactions`, 141
- `msg` (*eth.abc.ComputationAPI attribute*), 60
- `msg` (*eth.vm.computation.BaseComputation attribute*), 107
- `MuirGlacierBlock` (class in *eth.vm.forks.muir_glacier.blocks*), 139
- `MuirGlacierComputation` (class in *eth.vm.forks.muir_glacier.computation*), 139

MuirGlacierState (class in *eth.vm.forks.muir_glacier.state*), 141
MuirGlacierTransaction (class in *eth.vm.forks.muir_glacier.transactions*), 141
MuirGlacierUnsignedTransaction (class in *eth.vm.forks.muir_glacier.transactions*), 141
MuirGlacierVM (class in *eth.vm.forks.muir_glacier*), 142

N

name() (in module *eth.tools.builder.chain*), 100
new_access_list_transaction() (*eth.vm.forks.berlin.transactions.BerlinTransactionBuilder* class method), 150
new_dynamic_fee_transaction() (*eth.vm.forks.london.transactions.LondonTransactionBuilder* class method), 162
new_transaction() (*eth.vm.forks.berlin.transactions.BerlinTransactionBuilder* class method), 150
new_trie() (*eth.db.storage.StorageLookup* method), 92
new_unsigned_access_list_transaction() (*eth.vm.forks.berlin.transactions.BerlinTransactionBuilder* class method), 150
new_unsigned_dynamic_fee_transaction() (*eth.vm.forks.london.transactions.LondonTransactionBuilder* class method), 163
nonce (*eth.abc.BlockHeaderAPI* attribute), 45
nonce (*eth.abc.TransactionFieldsAPI* property), 47
nonce (*eth.abc.UnsignedTransactionAPI* attribute), 47
nonce (*eth.rlp.accounts.Account* property), 94
nonce (*eth.rlp.headers.BlockHeader* property), 95
nonce (*eth.rlp.transactions.BaseTransaction* property), 98
nonce (*eth.rlp.transactions.BaseTransactionFields* property), 97
nonce (*eth.rlp.transactions.BaseUnsignedTransaction* property), 98
nonce (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader* property), 166
nonce (*eth.vm.forks.arrow_glacier.transactions.ArrowGlacierLegacyTransaction* property), 168
nonce (*eth.vm.forks.arrow_glacier.transactions.ArrowGlacierUnsignedLegacyTransaction* property), 168
nonce (*eth.vm.forks.berlin.transactions.AccessListTransaction* property), 149
nonce (*eth.vm.forks.berlin.transactions.BerlinLegacyTransaction* property), 149
nonce (*eth.vm.forks.berlin.transactions.BerlinUnsignedLegacyTransaction* property), 151
nonce (*eth.vm.forks.berlin.transactions.TypedTransaction* property), 152
nonce (*eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction* property), 153
nonce (*eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader* property), 170
nonce (*eth.vm.forks.gray_glacier.transactions.GrayGlacierLegacyTransaction* property), 172
nonce (*eth.vm.forks.gray_glacier.transactions.GrayGlacierUnsignedLegacyTransaction* property), 172
nonce (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 156
nonce (*eth.vm.forks.london.transactions.DynamicFeeTransaction* property), 162
nonce (*eth.vm.forks.london.transactions.LondonLegacyTransaction* property), 162
nonce (*eth.vm.forks.london.transactions.LondonUnsignedLegacyTransaction* property), 163
nonce (*eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction* property), 164
nonce (*eth.vm.forks.muir_glacier.transactions.MuirGlacierTransaction* property), 141
nonce (*eth.vm.forks.muir_glacier.transactions.MuirGlacierUnsignedTransaction* property), 142
number (*eth.abc.BlockAPI* property), 49

O

Opcode (class in *eth.vm.opcode*), 110
OpcodeAPI (class in *eth.abc*), 54
opcodes (*eth.abc.ComputationAPI* attribute), 60
opcodes (*eth.vm.computation.BaseComputation* attribute), 107
opcodes (*eth.vm.forks.berlin.computation.BerlinComputation* attribute), 143
opcodes (*eth.vm.forks.byzantium.computation.ByzantiumComputation* attribute), 130
opcodes (*eth.vm.forks.constantinople.computation.ConstantinopleComputation* attribute), 133
opcodes (*eth.vm.forks.frontier.computation.FrontierComputation* attribute), 121
opcodes (*eth.vm.forks.homestead.computation.HomesteadComputation* attribute), 123
opcodes (*eth.vm.forks.istanbul.computation.IstanbulComputation* attribute), 137
opcodes (*eth.vm.forks.london.computation.LondonComputation* attribute), 138
opcodes (*eth.vm.forks.muir_glacier.computation.MuirGlacierComputation* attribute), 139
opcodes (*eth.vm.forks.petersburg.computation.PetersburgComputation* attribute), 135
opcodes (*eth.vm.forks.spurious_dragon.computation.SpuriousDragonComputation* attribute), 127
opcodes (*eth.vm.forks.tangerine_whistle.computation.TangerineWhistleComputation* attribute), 125
origin (*eth.abc.TransactionContextAPI* property), 54
origin (*eth.vm.transaction_context.BaseTransactionContext* property), 119
OutOfBoundsRead, 93

OutOfGas, 93

output (*eth.abc.ComputationAPI* property), 60

output (*eth.vm.computation.BaseComputation* property), 107

override_transaction_context() (*eth.abc.StateAPI* method), 66

override_transaction_context() (*eth.vm.state.BaseState* method), 117

P

pack_block() (*eth.abc.VirtualMachineAPI* method), 71

pack_block() (*eth.vm.base.VM* method), 113

parent_beacon_block_root (*eth.abc.BlockHeaderAPI* property), 45

parent_beacon_block_root (*eth.rlp.headers.BlockHeader* property), 95

parent_beacon_block_root (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 156

parent_hash (*eth.abc.MiningHeaderAPI* attribute), 44

parent_hash (*eth.rlp.headers.BlockHeader* property), 95

parent_hash (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader* property), 166

parent_hash (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader* property), 166

parent_hash (*eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader* property), 170

parent_hash (*eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader* property), 170

parent_hash (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 156

parent_hash (*eth.vm.forks.london.blocks.LondonMiningHeader* property), 156

ParentNotFound, 93

peek() (*eth.abc.CodeStreamAPI* method), 56

peek() (*eth.vm.code_stream.CodeStream* method), 108

pending_items() (*eth.db.diff.DBDiff* method), 87

pending_keys() (*eth.db.diff.DBDiff* method), 88

persist() (*eth.abc.AccountDatabaseAPI* method), 62

persist() (*eth.abc.AccountStorageDatabaseAPI* method), 61

persist() (*eth.abc.StateAPI* method), 66

persist() (*eth.db.account.AccountDB* method), 83

persist() (*eth.db.journal.JournalDB* method), 90

persist() (*eth.db.storage.AccountStorageDB* method), 91

persist() (*eth.vm.state.BaseState* method), 117

persist_block() (*eth.abc.ChainDatabaseAPI* method), 52

persist_block() (*eth.chains.base.Chain* method), 80

persist_block() (*eth.db.chain.ChainDB* method), 86

persist_checkpoint_header() (*eth.abc.HeaderDatabaseAPI* method), 50

persist_checkpoint_header() (*eth.db.header.HeaderDB* method), 89

persist_header() (*eth.abc.HeaderDatabaseAPI* method), 50

persist_header() (*eth.db.header.HeaderDB* method), 89

persist_header_chain() (*eth.abc.HeaderDatabaseAPI* method), 50

persist_header_chain() (*eth.db.header.HeaderDB* method), 89

persist_trie_data_dict() (*eth.abc.ChainDatabaseAPI* method), 52

persist_trie_data_dict() (*eth.db.chain.ChainDB* method), 87

persist_uncles() (*eth.abc.ChainDatabaseAPI* method), 52

persist_uncles() (*eth.db.chain.ChainDB* method), 87

persist_unexecuted_block() (*eth.abc.ChainDatabaseAPI* method), 52

persist_unexecuted_block() (*eth.db.chain.ChainDB* method), 87

PetersburgComputation (class in *eth.vm.forks.petersburg.computation*), 135

PetersburgState (class in *eth.vm.forks.petersburg.state*), 135

PetersburgVM (class in *eth.vm.forks.petersburg*), 135

pop1_any() (*eth.abc.StackAPI* method), 55

pop1_any() (*eth.vm.stack.Stack* method), 114

pop1_bytes() (*eth.abc.StackAPI* method), 55

pop1_bytes() (*eth.vm.stack.Stack* method), 114

pop1_int() (*eth.abc.StackAPI* method), 55

pop1_int() (*eth.vm.stack.Stack* method), 114

pop_any() (*eth.abc.StackAPI* method), 55

pop_any() (*eth.vm.stack.Stack* method), 114

pop_bytes() (*eth.abc.StackAPI* method), 55

pop_bytes() (*eth.vm.stack.Stack* method), 114

pop_ints() (*eth.abc.StackAPI* method), 55

pop_ints() (*eth.vm.stack.Stack* method), 114

pre_state() (in module *eth.tools.fixtures.fillers*), 103

precompiles (*eth.abc.ComputationAPI* property), 60

precompiles (*eth.vm.computation.BaseComputation* property), 107

prepare_child_message() (*eth.abc.ComputationAPI* method), 59

prepare_child_message() (*eth.vm.computation.BaseComputation* method), 106

prev_hashes (*eth.abc.ExecutionContextAPI* property), 57

prev_hashes (*eth.vm.execution_context.ExecutionContext* property), 108

previous_hashes (*eth.abc.VirtualMachineAPI* prop-

- erty*), 72
- `previous_hashes` (*eth.vm.base.VM* property), 114
- `program_counter` (*eth.abc.CodeStreamAPI* attribute), 56
- `program_counter` (*eth.vm.code_stream.CodeStream* attribute), 108
- `push_bytes()` (*eth.abc.StackAPI* method), 55
- `push_bytes()` (*eth.vm.stack.Stack* method), 115
- `push_int()` (*eth.abc.StackAPI* method), 55
- `push_int()` (*eth.vm.stack.Stack* method), 115
- `PyEVMError`, 93
- R**
- `r` (*eth.abc.TransactionFieldsAPI* property), 47
- `r` (*eth.rlp.transactions.BaseTransaction* property), 98
- `r` (*eth.rlp.transactions.BaseTransactionFields* property), 97
- `r` (*eth.vm.forks.arrow_glacier.transactions.ArrowGlacierLegacyTransaction* property), 168
- `r` (*eth.vm.forks.berlin.transactions.AccessListTransaction* property), 149
- `r` (*eth.vm.forks.berlin.transactions.BerlinLegacyTransaction* property), 149
- `r` (*eth.vm.forks.berlin.transactions.TypedTransaction* property), 152
- `r` (*eth.vm.forks.gray_glacier.transactions.GrayGlacierLegacyTransaction* property), 172
- `r` (*eth.vm.forks.london.transactions.DynamicFeeTransaction* property), 162
- `r` (*eth.vm.forks.london.transactions.LondonLegacyTransaction* property), 162
- `r` (*eth.vm.forks.muir_glacier.transactions.MuirGlacierTransaction* property), 141
- `raise_if_error()` (*eth.abc.ComputationAPI* method), 59
- `raise_if_error()` (*eth.vm.computation.BaseComputation* method), 106
- `read()` (*eth.abc.CodeStreamAPI* method), 56
- `read()` (*eth.abc.MemoryAPI* method), 54
- `read()` (*eth.vm.code_stream.CodeStream* method), 108
- `read()` (*eth.vm.memory.Memory* method), 109
- `read_bytes()` (*eth.abc.MemoryAPI* method), 54
- `read_bytes()` (*eth.vm.memory.Memory* method), 109
- `Receipt` (class in *eth.rlp.receipts*), 96
- `receipt_builder` (*eth.abc.BlockAPI* attribute), 49
- `receipt_builder` (*eth.vm.forks.berlin.blocks.BerlinBlock* attribute), 142
- `receipt_builder` (*eth.vm.forks.berlin.transactions.TypedTransaction* attribute), 151
- `receipt_builder` (*eth.vm.forks.london.blocks.LondonBlock* attribute), 154
- `receipt_builder` (*eth.vm.forks.london.transactions.LondonTypedTransaction* attribute), 163
- `receipt_root` (*eth.abc.MiningHeaderAPI* attribute), 45
- `receipt_root` (*eth.rlp.headers.BlockHeader* property), 96
- `receipt_root` (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader* property), 166
- `receipt_root` (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader* property), 166
- `receipt_root` (*eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader* property), 170
- `receipt_root` (*eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader* property), 170
- `receipt_root` (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 156
- `receipt_root` (*eth.vm.forks.london.blocks.LondonMiningHeader* property), 156
- `ReceiptAPI` (class in *eth.abc*), 46
- `ReceiptNotFound`, 93
- `record` (*eth.db.journal.JournalDB* attribute), 90
- `record()` (*eth.abc.AccountDatabaseAPI* method), 62
- `record()` (*eth.abc.AccountStorageDatabaseAPI* method), 61
- `record()` (*eth.db.account.AccountDB* method), 83
- `record()` (*eth.db.storage.AccountStorageDB* method), 91
- `refund_gas()` (*eth.abc.ComputationAPI* method), 59
- `refund_gas()` (*eth.abc.GasMeterAPI* method), 53
- `refund_gas()` (*eth.vm.computation.BaseComputation* method), 106
- `refund_gas()` (*eth.vm.gas_meter.GasMeter* method), 109
- `register_account_for_deletion()` (*eth.abc.ComputationAPI* method), 59
- `register_account_for_deletion()` (*eth.vm.computation.BaseComputation* method), 106
- `ReservedBytesInCode`, 93
- `reset()` (*eth.db.journal.JournalDB* method), 90
- `reset_cache()` (*eth.db.cache.CacheDB* method), 85
- `return_data` (*eth.abc.ComputationAPI* attribute), 60
- `return_data` (*eth.vm.computation.BaseComputation* attribute), 107
- `return_gas()` (*eth.abc.ComputationAPI* method), 59
- `return_gas()` (*eth.abc.GasMeterAPI* method), 53
- `return_gas()` (*eth.vm.computation.BaseComputation* method), 106
- `return_gas()` (*eth.vm.gas_meter.GasMeter* method), 109
- `Revert`, 93
- `revert()` (*eth.abc.StateAPI* method), 66
- `revert()` (*eth.vm.state.BaseState* method), 117
- `rlp_type` (*eth.vm.forks.berlin.receipts.TypedReceipt* attribute), 147
- `rlp_type` (*eth.vm.forks.berlin.transactions.TypedTransaction* attribute), 152
- `rollback_trie()` (*eth.db.storage.StorageLookup*

method), 92

S

s (*eth.abc.TransactionFieldsAPI* property), 47

s (*eth.rlp.transactions.BaseTransaction* property), 98

s (*eth.rlp.transactions.BaseTransactionFields* property), 97

s (*eth.vm.forks.arrow_glacier.transactions.ArrowGlacierLegacyTransaction* property), 168

s (*eth.vm.forks.berlin.transactions.AccessListTransaction* property), 149

s (*eth.vm.forks.berlin.transactions.BerlinLegacyTransaction* property), 149

s (*eth.vm.forks.berlin.transactions.TypedTransaction* property), 152

s (*eth.vm.forks.gray_glacier.transactions.GrayGlacierLegacyTransaction* property), 172

s (*eth.vm.forks.london.transactions.DynamicFeeTransaction* property), 162

s (*eth.vm.forks.london.transactions.LondonLegacyTransaction* property), 162

s (*eth.vm.forks.muir_glacier.transactions.MuirGlacierTransaction* property), 141

SchemaV1 (class in *eth.db.schema*), 90

seek() (*eth.abc.CodeStreamAPI* method), 56

seek() (*eth.vm.code_stream.CodeStream* method), 108

selfdestruct_eip2929() (in module *eth.vm.forks.berlin.logic*), 145

sender (*eth.abc.MessageAPI* attribute), 53

sender (*eth.abc.SignedTransactionAPI* property), 48

sender (*eth.vm.message.Message* attribute), 110

serialize() (*eth.vm.forks.berlin.receipts.BerlinReceiptBundle* class method), 146

serialize() (*eth.vm.forks.berlin.receipts.TypedReceipt* class method), 147

serialize() (*eth.vm.forks.berlin.transactions.BerlinTransactionBuilder* class method), 150

serialize() (*eth.vm.forks.berlin.transactions.TypedTransaction* class method), 152

serialize() (*eth.vm.forks.london.blocks.LondonBackwardsHeader* class method), 154

serialize() (*eth.vm.forks.london.headers.LondonBackwardsHeader* class method), 159

set() (*eth.abc.AccountStorageDatabaseAPI* method), 61

set() (*eth.abc.DatabaseAPI* method), 49

set() (*eth.db.backends.base.BaseDB* method), 84

set() (*eth.db.storage.AccountStorageDB* method), 91

set_balance() (*eth.abc.AccountDatabaseAPI* method), 63

set_balance() (*eth.abc.StateAPI* method), 66

set_balance() (*eth.db.account.AccountDB* method), 83

set_balance() (*eth.vm.state.BaseState* method), 117

set_block_transactions_and_withdrawals() (*eth.abc.VirtualMachineAPI* method), 71

set_block_transactions_and_withdrawals() (*eth.vm.base.VM* method), 113

set_code() (*eth.abc.AccountDatabaseAPI* method), 63

set_code() (*eth.abc.StateAPI* method), 66

set_code() (*eth.db.account.AccountDB* method), 83

set_code() (*eth.vm.state.BaseState* method), 117

set_header_timestamp() (*eth.abc.MiningChainAPI* method), 77

set_header_timestamp() (*eth.chains.base.MiningChain* method), 81

set_nonce() (*eth.abc.AccountDatabaseAPI* method), 63

set_nonce() (*eth.abc.StateAPI* method), 66

set_nonce() (*eth.db.account.AccountDB* method), 83

set_nonce() (*eth.vm.state.BaseState* method), 117

set_storage() (*eth.abc.AccountDatabaseAPI* method), 63

set_storage() (*eth.abc.StateAPI* method), 66

set_storage() (*eth.db.account.AccountDB* method), 83

set_storage() (*eth.vm.state.BaseState* method), 117

set_transient_storage() (*eth.abc.StateAPI* method), 66

set_transient_storage() (*eth.vm.state.BaseState* method), 117

setup_main_filler() (in module *eth.tools.fixtures.fillers.common*), 102

should_burn_gas (*eth.abc.ComputationAPI* property), 60

should_burn_gas (*eth.vm.computation.BaseComputation* property), 107

should_erase_return_data (*eth.abc.ComputationAPI* property), 60

should_erase_return_data (*eth.vm.computation.BaseComputation* property), 107

should_return_gas (*eth.abc.ComputationAPI* property), 60

should_return_gas (*eth.vm.computation.BaseComputation* property), 107

should_transfer_value (*eth.abc.MessageAPI* attribute), 53

should_transfer_value (*eth.vm.message.Message* attribute), 110

SignedTransactionAPI (class in *eth.abc*), 48

sload_eip2929() (in module *eth.vm.forks.berlin.logic*), 145

snapshot() (*eth.abc.StateAPI* method), 66

snapshot() (*eth.vm.state.BaseState* method), 117

SpuriousDragonComputation (class in *eth.vm.forks.spurious_dragon.computation*), 127

SpuriousDragonState (class in *eth.vm.forks.spurious_dragon.state*), 127

SpuriousDragonVM (class in *eth.vm.forks.spurious_dragon*), 127

[sstore_eip2929\(\)](#) (in [eth.vm.forks.berlin.logic](#)), 145
[sstore_eip2929_generic\(\)](#) (in [eth.vm.forks.berlin.logic](#)), 145
[Stack](#) (class in [eth.vm.stack](#)), 114
[stack_dup\(\)](#) ([eth.abc.ComputationAPI](#) method), 59
[stack_dup\(\)](#) ([eth.vm.computation.BaseComputation](#) method), 106
[stack_pop1_any](#) ([eth.vm.computation.BaseComputation](#) attribute), 107
[stack_pop1_any\(\)](#) ([eth.abc.StackManipulationAPI](#) method), 56
[stack_pop1_bytes](#) ([eth.vm.computation.BaseComputation](#) attribute), 107
[stack_pop1_bytes\(\)](#) ([eth.abc.StackManipulationAPI](#) method), 56
[stack_pop1_int](#) ([eth.vm.computation.BaseComputation](#) attribute), 107
[stack_pop1_int\(\)](#) ([eth.abc.StackManipulationAPI](#) method), 56
[stack_pop_any](#) ([eth.vm.computation.BaseComputation](#) attribute), 107
[stack_pop_any\(\)](#) ([eth.abc.StackManipulationAPI](#) method), 56
[stack_pop_bytes](#) ([eth.vm.computation.BaseComputation](#) attribute), 107
[stack_pop_bytes\(\)](#) ([eth.abc.StackManipulationAPI](#) method), 56
[stack_pop_ints](#) ([eth.vm.computation.BaseComputation](#) attribute), 107
[stack_pop_ints\(\)](#) ([eth.abc.StackManipulationAPI](#) method), 56
[stack_push_bytes](#) ([eth.vm.computation.BaseComputation](#) attribute), 107
[stack_push_bytes\(\)](#) ([eth.abc.StackManipulationAPI](#) method), 56
[stack_push_int](#) ([eth.vm.computation.BaseComputation](#) attribute), 107
[stack_push_int\(\)](#) ([eth.abc.StackManipulationAPI](#) method), 56
[stack_swap\(\)](#) ([eth.abc.ComputationAPI](#) method), 59
[stack_swap\(\)](#) ([eth.vm.computation.BaseComputation](#) method), 106
[StackAPI](#) (class in [eth.abc](#)), 55
[StackDepthLimit](#), 93
[StackManipulationAPI](#) (class in [eth.abc](#)), 56
[start_gas](#) ([eth.abc.GasMeterAPI](#) attribute), 53
[start_gas](#) ([eth.vm.gas_meter.GasMeter](#) attribute), 109
[state](#) ([eth.abc.ComputationAPI](#) attribute), 60
[state](#) ([eth.abc.VirtualMachineAPI](#) property), 72
[state](#) ([eth.vm.base.VM](#) property), 114
[state](#) ([eth.vm.computation.BaseComputation](#) attribute), 107
[state_root](#) ([eth.abc.AccountDatabaseAPI](#) property), 63
[state_root](#) ([eth.abc.MiningHeaderAPI](#) attribute), 45
[state_root](#) ([eth.abc.ReceiptAPI](#) property), 46
[state_root](#) ([eth.abc.StateAPI](#) property), 67
[state_root](#) ([eth.db.account.AccountDB](#) property), 84
[state_root](#) ([eth.rlp.headers.BlockHeader](#) property), 96
[state_root](#) ([eth.rlp.receipts.Receipt](#) property), 97
[state_root](#) ([eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader](#) property), 166
[state_root](#) ([eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader](#) property), 166
[state_root](#) ([eth.vm.forks.berlin.receipts.TypedReceipt](#) property), 147
[state_root](#) ([eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader](#) property), 170
[state_root](#) ([eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader](#) property), 170
[state_root](#) ([eth.vm.forks.london.blocks.LondonBlockHeader](#) property), 156
[state_root](#) ([eth.vm.forks.london.blocks.LondonMiningHeader](#) property), 156
[state_root](#) ([eth.vm.state.BaseState](#) property), 118
[StateAPI](#) (class in [eth.abc](#)), 64
[StateRootNotFound](#), 93
[StaticCallEIP2929](#) (class in [eth.vm.forks.berlin.logic](#)), 145
[storage_address](#) ([eth.abc.MessageAPI](#) property), 53
[storage_address](#) ([eth.vm.message.Message](#) property), 110
[storage_keys](#) ([eth.vm.forks.berlin.transactions.AccountAccesses](#) property), 149
[storage_root](#) ([eth.rlp.accounts.Account](#) property), 94
[StorageLookup](#) (class in [eth.db.storage](#)), 92
[support_dao_fork](#) ([eth.vm.forks.tangerine_whistle.TangerineWhistleVM](#) attribute), 125
[swap\(\)](#) ([eth.abc.StackAPI](#) method), 55
[swap\(\)](#) ([eth.vm.stack.Stack](#) method), 115

T

[TangerineWhistleComputation](#) (class in [eth.vm.forks.tangerine_whistle.computation](#)), 125
[TangerineWhistleState](#) (class in [eth.vm.forks.tangerine_whistle.state](#)), 125
[TangerineWhistleVM](#) (class in [eth.vm.forks.tangerine_whistle](#)), 125
[timestamp](#) ([eth.abc.ExecutionContextAPI](#) property), 57
[timestamp](#) ([eth.abc.MiningHeaderAPI](#) attribute), 45
[timestamp](#) ([eth.abc.StateAPI](#) property), 67
[timestamp](#) ([eth.rlp.headers.BlockHeader](#) property), 96
[timestamp](#) ([eth.vm.execution_context.ExecutionContext](#) property), 109
[timestamp](#) ([eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader](#) property), 166

[timestamp \(eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlock property\), 166](#)
[timestamp \(eth.vm.forks.gray_glacier.blocks.GrayGlacierBlock property\), 170](#)
[timestamp \(eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader property\), 171](#)
[timestamp \(eth.vm.forks.london.blocks.LondonBlockHeader property\), 156](#)
[timestamp \(eth.vm.forks.london.blocks.LondonMiningHeader property\), 156](#)
[timestamp \(eth.vm.state.BaseState property\), 118](#)
[to \(eth.abc.MessageAPI attribute\), 53](#)
[to \(eth.abc.TransactionFieldsAPI property\), 47](#)
[to \(eth.abc.UnsignedTransactionAPI attribute\), 47](#)
[to \(eth.rlp.transactions.BaseTransaction property\), 98](#)
[to \(eth.rlp.transactions.BaseTransactionFields property\), 97](#)
[to \(eth.rlp.transactions.BaseUnsignedTransaction property\), 98](#)
[to \(eth.vm.forks.arrow_glacier.transactions.ArrowGlacierLegacyTransaction property\), 168](#)
[to \(eth.vm.forks.arrow_glacier.transactions.ArrowGlacierUnsignedLegacyTransaction property\), 168](#)
[to \(eth.vm.forks.berlin.transactions.AccessListTransaction property\), 149](#)
[to \(eth.vm.forks.berlin.transactions.BerlinLegacyTransaction property\), 149](#)
[to \(eth.vm.forks.berlin.transactions.BerlinUnsignedLegacyTransaction property\), 151](#)
[to \(eth.vm.forks.berlin.transactions.TypedTransaction property\), 152](#)
[to \(eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction property\), 153](#)
[to \(eth.vm.forks.gray_glacier.transactions.GrayGlacierLegacyTransaction property\), 172](#)
[to \(eth.vm.forks.gray_glacier.transactions.GrayGlacierUnsignedLegacyTransaction property\), 172](#)
[to \(eth.vm.forks.london.transactions.DynamicFeeTransaction property\), 162](#)
[to \(eth.vm.forks.london.transactions.LondonLegacyTransaction property\), 162](#)
[to \(eth.vm.forks.london.transactions.LondonUnsignedLegacyTransaction property\), 163](#)
[to \(eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction property\), 164](#)
[to \(eth.vm.forks.muir_glacier.transactions.MuirGlacierTransaction property\), 141](#)
[to \(eth.vm.forks.muir_glacier.transactions.MuirGlacierUnsignedLegacyTransaction property\), 142](#)
[to \(eth.vm.message.Message attribute\), 110](#)
[topics \(eth.abc.LogAPI attribute\), 45](#)
[topics \(eth.rlp.logs.Log property\), 96](#)
[touch_account\(\) \(eth.abc.AccountDatabaseAPI method\), 63](#)

[touch_account\(\) \(eth.abc.StateAPI method\), 66](#)
[touch_account\(\) \(eth.db.account.AccountDB method\), 84](#)
[touch_account\(\) \(eth.vm.state.BaseState method\), 117](#)
[transaction_applied_hook\(\) \(eth.abc.VirtualMachineAPI method\), 71](#)
[transaction_builder \(eth.abc.BlockAPI attribute\), 49](#)
[transaction_builder \(eth.rlp.blocks.BaseBlock attribute\), 94](#)
[transaction_builder \(eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlock attribute\), 165](#)
[transaction_builder \(eth.vm.forks.berlin.blocks.BerlinBlock attribute\), 142](#)
[transaction_builder \(eth.vm.forks.gray_glacier.blocks.GrayGlacierBlock attribute\), 169](#)
[transaction_builder \(eth.vm.forks.london.blocks.LondonBlock attribute\), 154](#)
[transaction_builder \(eth.vm.forks.muir_glacier.blocks.MuirGlacierBlock attribute\), 139](#)
[transaction_context \(eth.abc.ComputationAPI attribute\), 60](#)
[transaction_context \(eth.vm.computation.BaseComputation attribute\), 107](#)
[transaction_context_class \(eth.abc.StateAPI attribute\), 67](#)
[transaction_context_class \(eth.vm.forks.frontier.state.FrontierState attribute\), 121](#)
[transaction_context_class \(eth.vm.state.BaseState attribute\), 118](#)
[transaction_executor_class \(eth.abc.StateAPI attribute\), 67](#)
[transaction_executor_class \(eth.vm.forks.arrow_glacier.state.ArrowGlacierState attribute\), 167](#)
[transaction_executor_class \(eth.vm.forks.berlin.state.BerlinState attribute\), 147](#)
[transaction_executor_class \(eth.vm.forks.frontier.state.FrontierState attribute\), 121](#)
[transaction_executor_class \(eth.vm.forks.gray_glacier.state.GrayGlacierState attribute\), 171](#)
[transaction_executor_class \(eth.vm.forks.london.state.LondonState attribute\), 160](#)
[transaction_executor_class](#)

- (*eth.vm.forks.spurious_dragon.state.SpuriousDragonState* attribute), 127
- transaction_executor_class* (*eth.vm.state.BaseState* attribute), 118
- transaction_root* (*eth.abc.MiningHeaderAPI* attribute), 45
- transaction_root* (*eth.rlp.headers.BlockHeader* property), 96
- transaction_root* (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader* property), 166
- transaction_root* (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader* property), 166
- transaction_root* (*eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader* property), 170
- transaction_root* (*eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader* property), 171
- transaction_root* (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 156
- transaction_root* (*eth.vm.forks.london.blocks.LondonMiningHeader* property), 156
- TransactionContextAPI* (class in *eth.abc*), 54
- TransactionExecutorAPI* (class in *eth.abc*), 63
- TransactionFieldsAPI* (class in *eth.abc*), 47
- TransactionNotFound*, 93
- transactions* (*eth.abc.BlockAPI* attribute), 49
- transactions* (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader* property), 165
- transactions* (*eth.vm.forks.berlin.blocks.BerlinBlockHeader* property), 142
- transactions* (*eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader* property), 169
- transactions* (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 154
- transactions* (*eth.vm.forks.muir_glacier.blocks.MuirGlacierBlockHeader* property), 139
- type_id* (*eth.abc.SignedTransactionAPI* attribute), 48
- type_id* (*eth.rlp.receipts.Receipt* attribute), 97
- type_id* (*eth.vm.forks.berlin.receipts.TypedReceipt* attribute), 147
- type_int* (*eth.exceptions.UnrecognizedTransactionType* property), 93
- typed_receipt_class* (*eth.vm.forks.berlin.receipts.BerlinReceiptBuilder* attribute), 146
- typed_receipt_class* (*eth.vm.forks.london.receipts.LondonReceiptBuilder* attribute), 159
- typed_transaction* (*eth.vm.forks.berlin.transactions.BerlinTransactionBuilder* attribute), 150
- typed_transaction* (*eth.vm.forks.london.transactions.LondonTransactionBuilder* attribute), 162
- TypedReceipt* (class in *eth.vm.forks.berlin.receipts*), 146
- TypedTransaction* (class in *eth.vm.forks.berlin.transactions*), 151
- ## U
- uncles* (*eth.abc.BlockAPI* attribute), 49
- uncles* (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader* property), 165
- uncles* (*eth.vm.forks.berlin.blocks.BerlinBlockHeader* property), 143
- uncles* (*eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader* property), 169
- uncles* (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 154
- uncles* (*eth.vm.forks.muir_glacier.blocks.MuirGlacierBlockHeader* property), 139
- uncles_hash* (*eth.abc.MiningHeaderAPI* attribute), 45
- uncles_hash* (*eth.rlp.headers.BlockHeader* property), 96
- uncles_hash* (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierBlockHeader* property), 166
- uncles_hash* (*eth.vm.forks.arrow_glacier.blocks.ArrowGlacierMiningHeader* property), 166
- uncles_hash* (*eth.vm.forks.gray_glacier.blocks.GrayGlacierBlockHeader* property), 170
- uncles_hash* (*eth.vm.forks.gray_glacier.blocks.GrayGlacierMiningHeader* property), 171
- uncles_hash* (*eth.vm.forks.london.blocks.LondonBlockHeader* property), 156
- uncles_hash* (*eth.vm.forks.london.blocks.LondonMiningHeader* property), 156
- UnrecognizedTransactionType*, 93
- UnsignedAccessListTransaction* (class in *eth.vm.forks.berlin.transactions*), 153
- UnsignedDynamicFeeTransaction* (class in *eth.vm.forks.london.transactions*), 163
- UnsignedTransactionAPI* (class in *eth.abc*), 47
- ## V
- v* (*eth.rlp.transactions.BaseTransaction* property), 98
- v* (*eth.rlp.transactions.BaseTransactionFields* property), 97
- v* (*eth.vm.forks.arrow_glacier.transactions.ArrowGlacierLegacyTransaction* property), 168
- v* (*eth.vm.forks.berlin.transactions.BerlinLegacyTransaction* property), 149
- v* (*eth.vm.forks.gray_glacier.transactions.GrayGlacierLegacyTransaction* property), 172
- v* (*eth.vm.forks.london.transactions.LondonLegacyTransaction* property), 162
- v* (*eth.vm.forks.muir_glacier.transactions.MuirGlacierTransaction* property), 141
- valid_positions* (*eth.vm.code_stream.CodeStream* attribute), 108
- validate()* (*eth.abc.BaseTransactionAPI* method), 46
- validate()* (*eth.abc.SignedTransactionAPI* method), 48

[validate\(\)](#) (*eth.rlp.transactions.BaseTransactionMethods* [validate_transaction\(\)](#) (*eth.abc.StateAPI* [method](#)), 97 [method](#)), 66
[validate\(\)](#) (*eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction* [validate_transaction\(\)](#) (*eth.abc.TransactionExecutorAPI* [method](#)), 153 [method](#)), 66
[validate\(\)](#) (*eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction* [validate_transaction\(\)](#) (*eth.abc.TransactionExecutorAPI* [method](#)), 164 [method](#)), 66
[validate_block\(\)](#) (*eth.abc.ChainAPI* [method](#)), 76 [validate_transaction\(\)](#) (*eth.vm.forks.frontier.state.FrontierState* [method](#)), 121
[validate_block\(\)](#) (*eth.abc.VirtualMachineAPI* [method](#)), 71 [validate_transaction\(\)](#) (*eth.vm.forks.homestead.state.HomesteadState* [method](#)), 123
[validate_block\(\)](#) (*eth.chains.base.Chain* [method](#)), 80 [validate_transaction\(\)](#) (*eth.vm.forks.london.state.LondonState* [method](#)), 160
[validate_block\(\)](#) (*eth.vm.base.VM* [method](#)), 113 [validate_transaction_against_header\(\)](#) (*eth.abc.VirtualMachineAPI* [method](#)), 71
[validate_chain\(\)](#) (*eth.abc.ChainAPI* [method](#)), 76 [validate_transaction_against_header\(\)](#) (*eth.vm.forks.frontier.FrontierVM* [method](#)), 120
[validate_chain\(\)](#) (*eth.chains.base.BaseChain* [method](#)), 77 [validate_uncle\(\)](#) (*eth.abc.VirtualMachineAPI* [class](#) [method](#)), 72
[validate_chain_extension\(\)](#) (*eth.abc.ChainAPI* [method](#)), 76 [validate_uncle\(\)](#) (*eth.vm.base.VM* [class](#) [method](#)), 113
[validate_chain_extension\(\)](#) (*eth.chains.base.BaseChain* [method](#)), 77 [validate_uncles\(\)](#) (*eth.abc.ChainAPI* [method](#)), 76
[validate_contract_code\(\)](#) (*eth.vm.forks.london.computation.LondonComputation* [class](#) [method](#)), 157 [validate_uncles\(\)](#) (*eth.chains.base.Chain* [method](#)), 80
[validate_contract_code\(\)](#) (*eth.vm.forks.spurious_dragon.computation.SpuriousDragonComputation* [class](#) [method](#)), 127 [value](#) (*eth.abc.TransactionFieldsAPI* [property](#)), 47
[validate_create_message\(\)](#) (*eth.vm.forks.spurious_dragon.computation.SpuriousDragonComputation* [class](#) [method](#)), 127 [value](#) (*eth.abc.UnsignedTransactionAPI* [attribute](#)), 47
[validate_gas\(\)](#) (*eth.vm.base.VM* [class](#) [method](#)), 113 [value](#) (*eth.rlp.transactions.BaseTransaction* [property](#)), 98
[validate_gas\(\)](#) (*eth.vm.forks.london.LondonVM* [class](#) [method](#)), 164 [value](#) (*eth.rlp.transactions.BaseTransactionFields* [property](#)), 97
[validate_header\(\)](#) (*eth.abc.VirtualMachineAPI* [class](#) [method](#)), 71 [value](#) (*eth.rlp.transactions.BaseUnsignedTransaction* [property](#)), 98
[validate_header\(\)](#) (*eth.vm.base.VM* [class](#) [method](#)), 113 [value](#) (*eth.vm.forks.arrow_glacier.transactions.ArrowGlacierLegacyTransaction* [property](#)), 168
[validate_london_normalized_transaction\(\)](#) (*in module eth.vm.forks.london.validation*), 164 [value](#) (*eth.vm.forks.arrow_glacier.transactions.ArrowGlacierUnsignedLegacyTransaction* [property](#)), 168
[validate_receipt\(\)](#) (*eth.abc.ChainAPI* [method](#)), 76 [value](#) (*eth.vm.forks.berlin.transactions.AccessListTransaction* [property](#)), 149
[validate_receipt\(\)](#) (*eth.abc.VirtualMachineAPI* [class](#) [method](#)), 71 [value](#) (*eth.vm.forks.berlin.transactions.BerlinLegacyTransaction* [property](#)), 149
[validate_receipt\(\)](#) (*eth.chains.base.Chain* [method](#)), 80 [value](#) (*eth.vm.forks.berlin.transactions.BerlinUnsignedLegacyTransaction* [property](#)), 151
[validate_receipt\(\)](#) (*eth.vm.base.VM* [class](#) [method](#)), 113 [value](#) (*eth.vm.forks.berlin.transactions.TypedTransaction* [property](#)), 152
[validate_receipt\(\)](#) (*eth.vm.forks.byzantium.ByzantiumVM* [class](#) [method](#)), 129 [value](#) (*eth.vm.forks.berlin.transactions.UnsignedAccessListTransaction* [property](#)), 153
[validate_seal\(\)](#) (*eth.abc.ChainAPI* [method](#)), 76 [value](#) (*eth.vm.forks.gray_glacier.transactions.GrayGlacierLegacyTransaction* [property](#)), 172
[validate_seal\(\)](#) (*eth.abc.VirtualMachineAPI* [method](#)), 71 [value](#) (*eth.vm.forks.gray_glacier.transactions.GrayGlacierUnsignedLegacyTransaction* [property](#)), 172
[validate_seal\(\)](#) (*eth.chains.base.Chain* [method](#)), 80 [value](#) (*eth.vm.forks.london.transactions.DynamicFeeTransaction* [property](#)), 162
[validate_seal\(\)](#) (*eth.vm.base.VM* [method](#)), 113
[validate_seal_extension\(\)](#) (*eth.abc.VirtualMachineAPI* [method](#)), 71
[validate_seal_extension\(\)](#) (*eth.vm.base.VM* [method](#)), 113

[value \(eth.vm.forks.london.transactions.LondonLegacyTransaction property\), 162](#)
[value \(eth.vm.forks.london.transactions.LondonUnsignedLegacyTransaction property\), 163](#)
[value \(eth.vm.forks.london.transactions.UnsignedDynamicFeeTransaction property\), 164](#)
[value \(eth.vm.forks.muir_glacier.transactions.MuirGlacierTransaction property\), 141](#)
[value \(eth.vm.forks.muir_glacier.transactions.MuirGlacierUnsignedTransaction property\), 142](#)
[value \(eth.vm.message.Message attribute\), 110](#)
[values \(eth.vm.stack.Stack attribute\), 115](#)
[VirtualMachineAPI \(class in eth.abc\), 67](#)
[VM \(class in eth.vm.base\), 111](#)
[vm_configuration \(eth.abc.ChainAPI attribute\), 76](#)
[vm_configuration \(eth.abc.HeaderChainAPI attribute\), 73](#)
[vm_configuration \(eth.chains.base.BaseChain attribute\), 77](#)
[VMError, 93](#)
[VMNotFound, 94](#)

W

[withdrawals \(eth.abc.BlockAPI attribute\), 49](#)
[withdrawals \(eth.vm.forks.muir_glacier.blocks.MuirGlacierBlock attribute\), 139](#)
[withdrawals_root \(eth.abc.MiningHeaderAPI property\), 45](#)
[withdrawals_root \(eth.rlp.headers.BlockHeader property\), 96](#)
[withdrawals_root \(eth.vm.forks.london.blocks.LondonBlockHeader property\), 156](#)
[wrapped_db \(eth.db.atomic.AtomicDB attribute\), 84](#)
[wrapped_db \(eth.db.batch.BatchDB attribute\), 85](#)
[write\(\) \(eth.abc.MemoryAPI method\), 54](#)
[write\(\) \(eth.vm.memory.Memory method\), 109](#)
[WriteProtection, 94](#)

Y

[y_parity \(eth.abc.SignedTransactionAPI property\), 48](#)
[y_parity \(eth.vm.forks.berlin.transactions.AccessListTransaction property\), 149](#)
[y_parity \(eth.vm.forks.berlin.transactions.TypedTransaction property\), 152](#)
[y_parity \(eth.vm.forks.london.transactions.DynamicFeeTransaction property\), 162](#)